

PARALLEL ALGORITHMS FOR BIOMECHANICAL OPTIMIZATION PROBLEMS

By

BYUNG IL KOH

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2005

BIOGRAPHICAL SKETCH

Byung Il Koh received the Bachelor of Science degree in electronics engineering from Ajou University, Suwon, Korea, in 1995. He worked as an assistant manager at the LG Information and Communications Co. in Seoul, Korea, from 1995 until 1999. He received the Master of Science degree in electrical and computer engineering from the University of Florida, Gainesville, FL, in 2001. He has been a graduate student pursuing the Doctor of Philosophy degree in electrical and computer engineering at the University of Florida since January 2002, and a research assistant at the High-performance Computing and Simulation (HCS) Research Laboratory since August 2000. His research was supported by the National Institutes of Health. After receiving his Ph.D. degree, he will be taking a senior researcher position at the Samsung Advanced Institute of Technology in Giheung, Korea.

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

PARALLEL ALGORITHMS FOR BIOMECHANICAL OPTIMIZATION PROBLEMS

By

Byung Il Koh

December 2005

Chair: Alan D. George

Cochair: Benjamin J. Fregly

Major Department: Electrical and Computer Engineering

The necessities of biomechanical computer simulation technologies in real-life medical practices have dramatically increased with high computational complexity, resulting in long computation time even on a modern high-end processors. As the complexity of musculoskeletal models continues to increase, so will the computational demands of biomechanical optimizations. For this reason, parallel computing for biomechanical optimizations is becoming more common. Our study introduces novel parallel algorithms for biomechanical optimization problems, and investigates their performance on a cluster of computers. The parallel algorithms are developed to satisfy computational requirements not readily supported by cutting-edge, high-performance, single-processor systems. By evaluating different parallel algorithms, an efficient parallel decomposition method is proposed for biomechanical optimization problems. An adaptive asynchronous parallel algorithm is proposed for the particle swarm global optimization method in order to overcome load imbalance problems that occur frequently

in parallel computing with a synchronous optimization approach. In addition, the proposed adaptive parallel algorithm is applied to a large-scale complex human movement prediction problem. The proposed parallel algorithms are implemented on scalable clusters of computers and performance issues are examined comparatively in terms of several critical factors such as parallelization and optimization methods. The performance results demonstrate that these parallel techniques may provide feasible and efficient solutions for human movement problems, which exhibit high computational cost.

PARALLEL ALGORITHMS FOR BIOMECHANICAL OPTIMIZATION PROBLEMS

By

BYUNG IL KOH

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2005

Copyright 2005

by

Byung Il Koh

This document is dedicated to the graduate students of the University of Florida.

ACKNOWLEDGMENTS

I am greatly indebted to Professor Alan D. George and Professor B.J. Fregly for their advice, guidance, and inspiration throughout my graduate education. I would like to thank Professor Raphael Haftka and Professor Renato Figueiredo for serving as members of my supervisory committee. I acknowledge and appreciate the support provided by the National Institutes of Health. I am grateful to Raj Subramaniyan, Jaco Schutte, Jeff Reinbolt, Kyusang Park, Kil Seok Cho and the other members of the High-performance Computing and Simulation (HCS) Research Laboratory and Computational Biomechanics Laboratory for their encouragement and invaluable help.

I thank my parents, parents-in-law, brothers, and sisters for their love, patience, and support. I also wish to acknowledge all my friends at the University of Florida and elsewhere. Most of all, I wish to dedicate all my work to my wife, Sun Hee, and my sons, Won Hee and Dae Hee.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
ABSTRACT	x
CHAPTER	
1 INTRODUCTION	1
2 BACKGROUND	6
2.1 Biomechanical Optimization Problems	6
2.2 Optimization Algorithms	9
2.3 Parallel Algorithms	11
2.3.1 Parallel programming	11
2.3.2 Parallel decomposition methods for biomechanical optimization	13
2.4 Parallel Performance Metrics	14
2.4.1 Execution time	14
2.4.2 Speedup	15
2.4.3 Efficiency	15
2.4.4 Other performance metrics	16
3 PARALLEL ALGORITHMS FOR SYSTEM IDENTIFICATION PROBLEM	17
3.1 Introduction	17
3.2 Computational Methodology	19
3.2.1 Description of gradient-based optimization algorithm	20
3.2.2 Description of analysis function	21
3.2.3 Description of decomposition methods	24
3.2.3.1 Gradient calculation decomposition	24
3.2.3.2 Analysis function decomposition	26
3.2.3.3 Combined decomposition	28
3.2.4 Description of evaluation metrics	30
3.3 Results	31
3.4 Discussion	33
3.5 Summary	37

4 PARALLEL ALGORITHMS FOR PARTICLE SWARM OPTIMIZATION	39
4.1 Introduction.....	39
4.2 Particle Swarm Optimization.....	41
4.2.1 Synchronous vs. asynchronous design	42
4.2.2 Parallel asynchronous design	44
4.3 Sample Optimization Problems	48
4.3.1 Analytical test problems	48
4.3.2 Biomechanical test problem	50
4.3.3 Evaluation metrics	52
4.4 Results.....	55
4.5 Discussion.....	60
4.6 Summary.....	62
5 EVALUATION OF PARALLEL GLOBAL SEARCH METHOD FOR LARGE- SCALE MOVEMENT OPTIMIZATION PROBLEMS.....	64
5.1 Introduction.....	64
5.2 Methodology	65
5.2.1 Parallel particle swarm optimization algorithms	66
5.2.2 Nonlinear least squares method.....	67
5.2.3 Large-scale biomechanical test problem	68
5.2.4 Optimization setup.....	71
5.3 Results.....	72
5.4 Discussion.....	75
5.5 Summary	76
6 CONCLUSIONS.....	78
APPENDIX EXAMPLE OF REFORMULATED FORWARD DYNAMIC OPTIMIZATION PROBLEM.....	82
BIOGRAPHICAL SKETCH.....	93

LIST OF TABLES

<u>Table</u>	<u>page</u>
3-1. Number of function evaluations and total execution time (wall-clock time) for sequential optimization	23
3-2. Optimizer, overhead, and computation time breakdown (in seconds) for the three parallel algorithms as a function of the number of processors.....	33
4-1. Summary of heterogeneous computing resources used to evaluate execution time for the biomechanical test problem.	53
4-2. Range of computation time delays [min, max] for each function evaluation of problem H3 with $n = 16$	54
4-3. Fraction of 100 optimization runs that successfully reached the correct solution to within the specified tolerance (see text) for each analytical test problem	55
4-4. Mean number of function evaluations required to reach the final solution for optimization runs that converged to the correct solution for each analytical test problem.....	57
4-5. Mean final cost function values and associated RMS marker distance and joint parameter errors after 10,000 function evaluations obtained with the parallel synchronous and asynchronous PSO algorithms	57
5-1. Mean for final fitness value and percent reduction on two peaks of the knee left knee abduction/adduction torque from the 5 runs of PAPSO and 1 run of LM-NLS method	73

LIST OF FIGURES

<u>Figure</u>	<u>page</u>
2-1. Components commonly included in a multi-joint model of movement.....	7
2-2. Block diagram of inverse dynamic analysis of a multi-joint model of movement.....	8
2-3. Block diagram of forward dynamic analysis of a multi-joint model of movement ...	8
3-1. Kinematic ankle joint model used in analysis function.....	22
3-2. Block diagram for gradient calculation decomposition in a 4-processor system. Parallelization is performed only for the optimizer gradient calculations	25
3-3. Block diagram for analysis function decomposition in a 5-processor system. Parallelization is performed only for the analysis function.....	27
3-4. Block diagram for combined decomposition in a 10-processor system.....	29
3-5. Performance metrics for the parallel decomposition methods as a function of the number of processors	32
4-1. Pseudo-code for a sequential synchronous PSO algorithm.....	43
4-2. Pseudo-code for a sequential asynchronous PSO algorithm	44
4-3. Block diagrams for (a) parallel asynchronous and (b) parallel synchronous PSO algorithms.....	45
4-4. Block diagram for first-in-first-out centralized task queue with p particles on a k - processor system	47
4-5. Kinematic ankle joint model used in the biomechanical test problem	51
4-6. Parallel efficiency for the parallel asynchronous (circles) and parallel synchronous (triangles) PSO algorithms as a function of number of processors, computational delay time (0.5, 1, or 2 seconds), and computational delay time variation (0, 20, or 50%)	56
4-7. Mean convergence history plots over 10,000 function evaluations for the biomechanical test problem obtained with the parallel asynchronous and parallel synchronous PSO algorithms	57

4-8. Execution time (a), speedup (b), and parallel efficiency (c) for parallel asynchronous and parallel synchronous PSO algorithms for the biomechanical test problem in a homogeneous computing environment.....	59
4-9. Execution time for parallel asynchronous and parallel synchronous PSO algorithms for the biomechanical test problem in a heterogeneous computing environment.....	60
5-1. Schematic of the 27 degree-of-freedom full-body gait model used to predict novel gait motions that reduce the peak knee adduction torque.....	69
5-2. Left knee abduction/adduction (A/A) torque curves calculated from the experimental gait data (solid line) and predicted by two optimizations: PAPSO and nonlinear least squares method (LM-NLS)	73
5-3. Left center of pressure (CoP) trajectories in pelvis progression frame	74

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

PARALLEL ALGORITHMS FOR BIOMECHANICAL OPTIMIZATION PROBLEMS

By

Byung Il Koh

December 2005

Chair: Alan D. George

Cochair: Benjamin J. Fregly

Major Department: Electrical and Computer Engineering

The necessities of biomechanical computer simulation technologies in real-life medical practices have dramatically increased with high computational complexity, resulting in long computation time even on a modern high-end processors. As the complexity of musculoskeletal models continues to increase, so will the computational demands of biomechanical optimizations. For this reason, parallel computing for biomechanical optimizations is becoming more common. Our study introduces novel parallel algorithms for biomechanical optimization problems, and investigates their performance on a cluster of computers. The parallel algorithms are developed to satisfy computational requirements not readily supported by cutting-edge, high-performance, single-processor systems. By evaluating different parallel algorithms, an efficient parallel decomposition method is proposed for biomechanical optimization problems. An adaptive asynchronous parallel algorithm is proposed for the particle swarm global optimization method in order to overcome load imbalance problems that occur frequently

in parallel computing with a synchronous optimization approach. In addition, the proposed adaptive parallel algorithm is applied to a large-scale complex human movement prediction problem. The proposed parallel algorithms are implemented on scalable clusters of computers and performance issues are examined comparatively in terms of several critical factors such as parallelization and optimization methods. The performance results demonstrate that these parallel techniques may provide feasible and efficient solutions for human movement problems, which exhibit high computational cost.

CHAPTER 1 INTRODUCTION

Computational biomechanics is the study of the dynamics of the human movement system using computer simulation technologies. With the rapid development of high-speed computers over the past decades and the difficulty of measuring movement-related quantities of the human body that cannot be measured experimentally, the possibilities and necessities of exploiting computer simulation technologies in real-life medical practices have dramatically increased. With these computational simulation technologies, many researchers in the biomechanical arena can predict the dynamics of human movement employing the complex musculoskeletal system. The results of such simulations can be used in many applications such as orthopedic treatment or surgery.

Biomechanical simulations frequently use optimization methods because of the indeterminate characteristic of the human movement system. The indeterminacy of the human movement system is because there are typically more muscles than degrees of freedom (DOFs) in the model [1]. Although the torques at the joints are the primary forces in movement, muscles and ligaments attached to the bones also contribute to the action. Several different categories of biomechanical optimization problems have been solved to predict the movement-related quantities. Forward dynamic, inverse dynamic, and inverse static optimizations have been used to predict muscle, ligament, and joint contact forces during experimental or predicted movements. System identification optimizations have been employed to tune a variety of musculoskeletal model parameters to experimental data. Image matching optimizations have been performed to align

implant and bone models in vivo fluoroscopic images collected during loaded functional activities.

Several optimization algorithms have been used for these biomechanical optimization problems. Gradient-based, simplex, simulated annealing, genetic, and particle swarm algorithms have been used for such applications. Since these optimization algorithms iteratively evaluate the cost function and/or constraints until the solution reaches the optimum, they involve high computational cost. Moreover, for large-scale problems, these optimization algorithms exhibit even higher computational cost. As the complexity of biomechanical systems increases, the computational cost increases dramatically. Even with fast-converging optimization algorithms, the complexities of present-day biomechanical models can require thousands of function evaluations to achieve convergence [2,3]. Although the performance of single-processor computers has vastly increased in recent years, computation time can still be a severe limiting factor.

To circumvent this limitation, the computational load of biomechanical optimization problems can be decomposed and distributed to multiple processors in a parallel computer system. Three general approaches are possible to develop parallel algorithms for general optimization [4]. The first approach is to decompose the design space so that the optimization processes are parallelized at a high level: multiple evaluations of the analysis function (or objective function or cost function) and/or constraints can be distributed to the processors. This approach is called a parallel optimization algorithm. The second approach is to parallelize the individual evaluations of the objective function and/or constraints so that the decomposition is performed within the analysis function itself. The third approach is to parallelize the linear algebra

involved in each iteration. This method is based on the assumption that the linear algebraic calculations of the optimization algorithm occupy most of the computation time. However, the last decomposition method is not useful to biomechanical optimization problems, since most of the computation time is spent on evaluation of the analysis function. In addition to these three methods, multi-level decomposition, which decomposes the high level as an optimization process and decomposes the low level as an analysis function, can be another option.

Among these parallel decomposition methods, the performance of parallel optimization algorithms frequently suffers from load imbalance caused by several factors such as user load, heterogeneity of processors, and load variation in run time. An asynchronous parallel optimization algorithm considered in this research can resolve this performance limitation. Most of the parallel optimization algorithms in use today synchronously update the search direction, which requires every processor to wait for the responses from all other processors at every iteration. However, the asynchronous approach, which updates the search direction without any synchronization points, can overcome these limitations.

In the first part of this dissertation, three types of parallel decomposition methods that can be used in biomechanical optimization problems are presented and their parallel performance is evaluated on a cluster of Linux servers. By decomposing and mapping the workloads over multiple processors, the total execution time to solve optimization can be dramatically reduced. The parallel algorithms are based on domain decomposition methods. The first method decomposes the design space, the second decomposes the time domain, and the last method simultaneously decomposes the design space and time

domain. The performance of the algorithms is analyzed in terms of execution time, speedup, and parallel efficiency to demonstrate that these parallel algorithms enhance the computational performance of the biomechanical optimization problem.

In the second part of this dissertation, a novel adaptive parallel algorithm for global optimization method is proposed. An asynchronous parallel optimization algorithm is developed and evaluated on a cluster with varied environment factors such as heterogeneity in computing environment and computational cost. The proposed asynchronous approach is based on the particle swarm optimization (PSO) algorithm and works to overcome the load imbalance problem. The performance of the proposed algorithm is analyzed in terms of optimization robustness, performance, and parallel performance using commonly used small- to medium-scale analytical problems and a medium-scale biomechanical system identification problem that generates load variation in run time.

Finally, the adaptive parallel global search algorithm is applied to a real clinical problem. The problem involves a large-scale biomechanical movement optimization with a complex musculoskeletal model. Generally, global search methods have not been applied to large-scale biomechanical optimization problems because of computational complexity and presence of many local minima. The performance of optimization is compared to that of a commonly used gradient-based optimization algorithm. In addition to optimization performance analysis, the clinical issues of gait modifications are evaluated and presented in this study.

The remainder of this dissertation is organized as follows. In Chapter 2, background on basic biomechanical optimization problems, optimization algorithms, and

parallel computing concepts is provided. In Chapter 3, novel parallel algorithms for biomechanical optimization problems are presented and evaluated. The parallel asynchronous particle swarm optimization algorithm and its associated performance analysis are presented in Chapter 4. Chapter 5 presents an evaluation of parallel global search methods for large-scale movement optimization problem. Chapter 6 summarizes key points of insight, and directions for future research.

CHAPTER 2 BACKGROUND

In this chapter, the basic concepts of biomechanical dynamic simulations and optimization algorithms are presented to provide a background for concrete understanding of biomechanical optimization problems. Parallel programming and its decomposition methods related to this research are also described in this chapter. In addition, parallel performance metrics, which are important to analyze the parallel performance, are presented.

2.1 Biomechanical Optimization Problems

Computational biomechanics is the study of the dynamics that use computer modeling and simulation technologies. There are two reasons that computer modeling and simulation of human movement studies have increased in number in recent years. First, this method clearly explains how movement is produced by the combined dynamics of the musculoskeletal system. Second, the ever-increasing performance of computers now enables biomechanics researchers to perform simulations in a reasonable amount of time. Two dynamic optimization methods are commonly used to perform the computer modeling and simulation of human movement. In this section, the concepts of two dynamic optimization methods are described.

The movement simulation is composed of several steps (Figure 2-1) [6,7]. When muscle excitations are input to the system, muscle excitation-contraction dynamics produce muscle activations that are input to musculotendon dynamics. Then the muscle forces from the musculotendon dynamics are input to skeletal dynamics to generate the

body motion. A detailed explanation of the diagram can be seen in Anderson and Pandy [7].

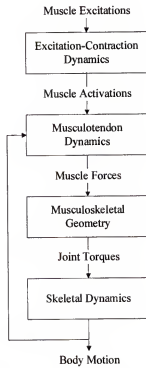


Figure 2-1. Components commonly included in a multi-joint model of movement.

Movement simulations can be performed to determine the quantities that cannot be measured noninvasively using two different approaches. The inverse dynamics method uses a noninvasive measurement of body motion (position, velocity, and acceleration of each segment) and external forces to calculate the muscle forces (Figure 2-2) [8]. Meanwhile, the forward dynamics method uses muscle excitation or activation as inputs to calculate the corresponding body motion (Figure 2-3) [8]. Most of the problems that use these dynamics methods are based on optimization theory. Since the number of muscles crossing a joint is greater than the number of degrees of freedom specifying joint movement, the force developed by each muscle cannot be uniquely determined.

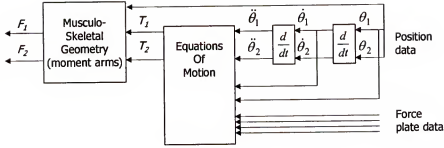


Figure 2-2. Block diagram of inverse dynamic analysis of a multi-joint model of movement.

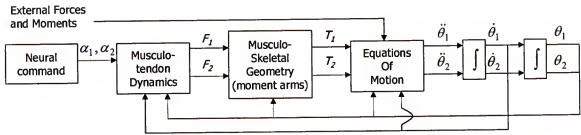


Figure 2-3. Block diagram of forward dynamic analysis of a multi-joint model of movement.

The inverse dynamics optimization method takes the motion of the model at discrete time steps during the execution of the task as independent variables and obtains solutions for the joint torques by means of inverse dynamics. To calculate the muscle forces, some means of static optimization has to be effected to obtain the muscle forces from the calculated joint torques because of the redundancy system in the body. This method is inherently computationally less expensive than the forward dynamics optimization method because it solves a static optimization problem for each time step. Meanwhile, the forward dynamics optimization method has to optimize for one complete movement cycle. This is the main difference between these two methods and is also the reason that forward dynamics optimization solutions are more computationally expensive.

However, the inverse dynamics optimization method has several drawbacks, which include the following [6]:

- Large errors may be present in the resulting joint torques and the muscle forces derived from the method because of the difficulty in estimating accurate velocity and accelerations from experimental position data, which contains significant errors in itself.
- It is difficult to incorporate muscle physiology into the formulation of a static optimization problem because estimates of muscle length and contraction velocity depend on the accuracy with which the positions and velocities of the body segments can be measured.
- It is an analysis-based approach: a model of the goal of the motor task cannot be included in the formulation of this problem.

Even though the forward dynamics optimization is more computationally expensive, it is more powerful than the inverse dynamics optimization method for several reasons. Muscle physiology can be easily incorporated in the formulation of the problem, since the system equations are integrated forward in time. A model of the goal of the motor task can also be included because the optimization is performed over a complete cycle of the task.

2.2 Optimization Algorithms

Optimization algorithms are widely used to solve various engineering problems in which the problem formulation includes the indeterminate system. Since the indeterminate system has a greater number of variables than that of equations, numerous solutions exist. Thus, there is a need to use optimization algorithms to find an optimum solution which is close to the user's interests. Optimization problems are translated to mathematical form by defining several elements [9]. The first is the design variables, which are variables that the users can change to optimize the solution. The second is the objective functions which are figures of merit to be minimized or maximized. The third

is the constraint functions, which are the limits that must be satisfied. The following equations form the standard optimization problem.

Find a set of design variables, $X_i, i = 1, N$ contained in vector X that will

$$\text{Minimize } F(X) \quad (2-1)$$

$$\text{Subject to: } h_i(X) = 0, \quad i = 1, \dots, M \quad (2-2)$$

$$g_j(X) \leq 0, \quad j = 1, \dots, L \quad (2-3)$$

$$X_i^L \leq X_i \leq X_i^U, \quad i = 1, \dots, N \quad (2-4)$$

Equation 2-1 defines the objective function, $F(X)$, which depends on the values of the design variables. Equations 2-2 and 2-3 define equality and inequality constraints respectively. Constraint functions, $h_i(X)$ and $g_j(X)$, are required to satisfy during the optimization. Equation 2.4 specifies the region of search for the minimum. The lower and upper bounds for design variable of X_i are shown as X_i^L and X_i^U respectively. The bounds defined by Equation 2-4 are referred to as side constraints. A clear understanding of the generality of this formulation makes the breadth of problems that can be addressed apparent.

Generally, optimization algorithms can be classified in two different categories: gradient-based and non-gradient-based optimization algorithms. A gradient-based optimization algorithm usually uses finite difference (or analytical) gradient calculations to determine the search directions. Gradient-based optimization is commonly used in biomechanical optimization problems because of its fast convergence characteristic. However, this algorithm has several drawbacks [10]. First, the algorithm tends to determine a local rather than global minimum. Second, it is sensitive to initial guess so that multiple runs should be performed. Thirdly, it is also sensitive to design variable

scaling and finite difference step size when the numerical derivatives are performed. Meanwhile, non-gradient-based optimization uses its own strategy to determine the search region. Most of the non-gradient-based optimization algorithms determine a global minimum, but the computation time to find the optimum is much larger than that of gradient-based optimization [3].

Since biomechanical optimization problems are typically nonlinear in design variables, gradient-based nonlinear programming has been widely used. The optimization algorithms that have so far been used in biomechanical optimization problems can be found in Schutte et al. [10]. The details of specific optimization algorithms that will be used in this research are described in each phase of the research.

2.3 Parallel Algorithms

2.3.1 Parallel programming

Parallel programming is a method to execute large, complex tasks faster by decomposing the tasks into smaller simultaneously performed tasks. The job of creating a parallel program from a sequential one consists of four steps [11]. The first is to decompose the sequential algorithm or data into smaller tasks. The second is to assign the tasks to processes. The process is an abstract entity that performs tasks. The third is to orchestrate the necessary data access, communication, and synchronization. The last step is to map or bind the processes to processors.

There are three methods for decomposing a problem into smaller tasks: functional decomposition (function parallelism), domain decomposition (data parallelism), or a combination of both [11]. The functional decomposition method decomposes the problem into different tasks, which can be distributed to multiple processors for simultaneous execution. Entirely different calculations can be performed concurrently on

either the same or different data. This method is good to use when there is no static structure or fixed determination of number of calculations to be performed. Domain decomposition is decomposing the problem's data domain and distributing portions to multiple processors for simultaneous execution. This method is good for problems where data is static, such as factoring and solving large matrix or finite difference calculations. Other problems that could use this method include large multi-body dynamics and fluid dynamics. In addition to these two methods, both function parallelism and data parallelism are often available together in an application and provide a hierarchy of levels of parallelism [11].

In addition to decomposition methods, concrete understanding of the inter-processor communications of the parallel program is essential. Generally, message passing communication is programmed explicitly. The programmer must understand and code the communication in a parallel program. In an ideal environment, the parallel compiler and run-time systems generate the parallel program implicitly from a sequential program. The compiler determines all of the inherent algorithmic parallelism and communication necessary to execute the program in parallel. However, the implicit approach is not usually as efficient as explicit parallelization. To achieve good performance from the parallel program, programmers need to explicitly write their own algorithm using the best communication possible.

The message-passing interface (MPI), which is a standard portable message-passing library definition of a core set of parallel communication functions [12,13], is used in this research for the message-passing programming model on a cluster of computers. MPI is available to both Fortran and C programs, which could be used in a

wide variety of parallel machines. All parallelism is explicit, so the programmer is responsible for parallelizing the program and implementing MPI constructs. This explicit approach to derive parallel programs allows the programmer to determine exactly where and how much communication and synchronization will take place. This capability is important in the design of new parallel programs because it gives the user the flexibility required to find the most efficient parallelization technique. The explicit approach over a cluster of computers efficiently supports the coarse- and medium-grained decomposition of biomechanical optimization problems. Fine-grained decomposition is not recommended in this parallel model due to significant inter-processor communication overhead. Details of communication functions can be found in Snir et al. [12] and the Message Passing Interface Forum [13].

2.3.2 Parallel decomposition methods for biomechanical optimization

Parallel decomposition is of interest in the optimization arena because of the expensive computational cost inherent in solving optimization problems. As described earlier, all optimization algorithms are iterative. Each iteration involves at least one evaluation of analysis function and/or one evaluation of constraint, and some linear algebraic computation. The evaluation of analysis function, constraints, and linear algebraic computations can be the primary expenses of optimization. In addition, the many iterations required to reach a converged solution can also be one of the main expenses.

By evaluating these optimization characteristics, general decomposition methods for optimization problems can be divided into three different categories [4]. First, multiple evaluations of the analysis function and/or constraints can be distributed to the processors. Parallel optimization algorithms belong in this category. Gradient

evaluations can be parallelized using either finite difference or analytical calculation approach in gradient-based optimization, and evaluations of samples through the design space can be parallelized in non-gradient-based optimization. Since this decomposition method is easy to develop and inherently parallelizable, most biomechanical optimization researchers use this parallel approach. Second, the individual evaluations of analysis functions can be parallelized. When the evaluation of the analysis function is the most expensive, this method is the most efficient. Biomechanical optimization problems could benefit from this decomposition method. Unfortunately, no research has been performed in the biomechanical optimization area using this decomposition method. Third, the linear algebraic computation involved in the optimization algorithm can be parallelized in each iteration. This decomposition method is not of interest to biomechanical researchers, since most of the computation time in biomechanical optimization problems is placed into the evaluations of the analysis function.

2.4 Parallel Performance Metrics

Several performance metrics are important in analyzing the performance of a parallel program. The metrics introduced in the following section can be used to compare sequential and parallel programs.

2.4.1 Execution time

Execution time is defined as the time elapsed from the beginning of a program execution to the end of the program execution. Execution time is an essential factor for determining the response time of the system. The parallel execution time, T_p , is composed of three parts: computation time, T_{comp} , communication time, T_{comm} , and time to synchronize between processors, T_{sync} (Equation 2-5).

$$T_p = T_{comp} + T_{comm} + T_{sync} \quad (2-5)$$

2.4.2 Speedup

Speedup describes how much faster a parallel program executes as compared to the sequential program and is defined as Equation 2-6.

$$S = \frac{T_s}{T_p} \quad (2-6)$$

where T_s is execution time of the best available sequential program and T_p is execution time of the parallel program on a collection of processors [14]. The larger the speedup is, the better the quality of the parallel algorithm. Amdahl's law limits the speedup to the amount of inherent parallelism found in the algorithm [15]. Amdahl's law states that

$$S_N = \frac{T_s}{T_p} = \frac{T_s}{\alpha T_s + \frac{(1-\alpha)T_s}{N}} \quad (2-7)$$

where S_N is speedup for an N -processor system, and α is the fraction of the algorithm that is not parallelizable. In most cases, ideal speedup, which is the number of processors, is the ultimate goal when parallelizing a sequential program.

2.4.3 Efficiency

The parallel efficiency, E , is defined as the speedup divided by the number of processors used. The efficiency is highest when all processors are used throughout the entire execution of the program, and lowest efficiency occurs when the program runs almost exclusively on a single processor or the communication overhead dwarfs the computation time of each process.

$$E = \frac{S_N}{N} \quad (2-8)$$

2.4.4 Other performance metrics

In addition to the performance metrics addressed above, the performance of the parallel programs can be measured by other aspects, such as memory requirements.

Memory requirement is defined as the minimum amount of memory space required to run a parallel program.

CHAPTER 3

PARALLEL ALGORITHMS FOR SYSTEM IDENTIFICATION PROBLEM

In this chapter, we present three parallel algorithms based on domain decomposition techniques to meet intensive computational requirements for biomechanical optimization problems using gradient-based optimization methods. Section 3.1 addresses the overview of this chapter. Section 3.2 describes the computational methodology with background on the gradient-based optimization algorithm and problem formulation as well as the computational tasks of the sequential algorithm. It also includes the features of three parallel algorithms for the biomechanical system identification problem. Section 3.3 presents experimental results for three parallel algorithms, and their performance is discussed in Section 3.4. Finally, Section 3.5 presents a brief summary.

3.1 Introduction

Optimization algorithms are frequently used to solve system identification or movement prediction problems utilizing complex musculoskeletal models [6,15-19]. To date, gradient-based, simplex, simulated annealing, genetic, and particle swarm algorithms have been used for such applications [2,3,6,15-19]. For large-scale problems, these algorithms have a high computational cost since they iteratively evaluate the cost function and constraints to obtain a converged solution. Moreover, as the complexity of biomechanical systems increases (e.g., increased number of body segments, degrees of freedom, or controlled muscle forces), the computational expense of simulating the human musculoskeletal system increases dramatically [18]. Even with fast-converging

optimization algorithms, the complexities of present-day biomechanical models can require thousands of function evaluations to achieve convergence [2,3]. Although the performance of single-processor computers has vastly increased in recent years, computation time can still be a limiting factor.

To circumvent this limitation, the computational load of biomechanical optimization problems can be decomposed and distributed to multiple processors in a parallel computer system. Two general approaches are possible to develop parallel algorithms for biomechanical optimization. The first takes advantage of data independence, an inherent characteristic of many optimization algorithms. Most optimization algorithms iteratively evaluate an analysis function (i.e., the cost function and constraints) to produce a converged solution. Some of these function evaluations have data-independent characteristics that can be parallelized. For example, research has been performed on gradient-based parallel optimization algorithms that seek to parallelize function evaluations for finite-difference gradient calculations [6,18-19]. Similarly, sampling within the search space by non-gradient-based algorithms has also been parallelized [3,10,20-22]. The second approach seeks to parallelize the analysis function itself so that the workload is distributed to the processors with finer granularity. Together, the optimization algorithm and analysis function can be viewed as the upper and lower levels, respectively, of a two-level process, where either level can potentially be parallelized. To date, no biomechanical studies have sought to parallelize the lower level or both levels simultaneously.

This chapter describes three parallel algorithms for biomechanical optimization and presents performance evaluations based on level of parallelization: upper level, lower

level, and both levels. The concepts are demonstrated using a biomechanical system identification problem for a kinematic ankle joint model. Joint positions and orientations in the body segments that result in the best match to experimental movement data are determined using an unconstrained gradient-based optimization algorithm. The primary emphasis of this work is to demonstrate that lower-level parallelization can be one way to solve biomechanical optimization problems in a reasonable amount of time. Furthermore, it can be the best way to overcome the computational limitations of existing upper-level parallel optimization algorithms. We also show that two-level parallelization can use more available resources in parallel computer systems, even when the optimization problem has a small number of design variables.

3.2 Computational Methodology

Parallel algorithms seek to distribute the workload evenly across available processors and then gather the results with minimal overhead. The overhead can be communication, synchronization, and other overhead caused by algorithm decomposition. The workload distribution determines the class of parallel algorithm in terms of granularity: coarse-grained or fine-grained. Coarse-grained parallel algorithms have less overhead than do fine-grained algorithms but at the cost of frequent load imbalance. By contrast, fine-grained parallel algorithms are more evenly distributed and balanced but generally suffer from higher communication and synchronization overhead.

In this research, three parallel algorithms are developed: one coarse-grained, one fine-grained and one medium-grained. The coarse-grained parallel algorithm decomposes an upper-level gradient-based optimization algorithm. The fine-grained parallel algorithm decomposes a lower-level analysis function. The medium-grained

parallel algorithm decomposes both levels simultaneously (combined approach of first two parallel algorithms).

The following sections describe the gradient-based optimization algorithm, an ankle joint system identification problem with corresponding analysis function, and the decomposition methods for the three parallel algorithms. A thorough understanding of the optimization algorithm and analysis function is necessary to develop parallel decomposition methods, since parallel algorithms depend heavily on the structure of the algorithms being decomposed. Each parallel algorithm is developed to achieve an evenly distributed workload.

3.2.1 Description of gradient-based optimization algorithm

An unconstrained gradient-based optimization algorithm (Broydon-Fletcher-Goldfarb-Shanno, or BFGS) available in commercial software (VisualDOC, VanderPlaats R&D, Colorado Springs, CO) [23] was used for the upper-level optimization in this study. The BFGS algorithm is a Quasi-Newton method that creates an approximation of the inverse Hessian matrix. The Hessian matrix determines the direction used by the algorithm for line searches. The search direction S_k and updated design variables X_k are defined by Equation 3-1 and 3-2 respectively [23]:

$$S_k = -H^{-1}\nabla F(X_{k-1}) \quad (3-1)$$

$$X_k = X_{k-1} + \alpha_k S_k \quad (3-2)$$

where α_k is chosen to minimize $F(X_{k-1} + \alpha_k S_k)$, H the Hessian matrix, X the vector of design variables, $F(X)$ a function evaluation, and subscript k the iteration number. The Hessian matrix is updated by information from line searches and gradients calculations at each iteration [23].

The BFGS algorithm consists of three steps. First, the optimizer creates an approximation of the inverse Hessian matrix, where initially the Hessian matrix is set to the identity matrix. Second, the optimizer calculates gradients to determine the search direction. Third, the optimizer performs a line search to determine how far it can proceed along the search direction. The second and third steps determine the search direction and line search step size, respectively, through function evaluations. The function evaluations for gradient calculations are inherently parallelizable, since no data dependencies are involved.

3.2.2 Description of analysis function

A biomechanical system identification problem was used to evaluate three parallel decomposition methods. The problem involves determination of patient-specific parameter values that permit a kinematic ankle joint model to reproduce experimental movement data as closely as possible [20]. Twelve parameters (treated as design variables) specify the fixed positions and orientations of joint axes in adjacent body segments (i.e., shank, talus, and foot) within the three-dimensional, eight degree-of-freedom (DOF) kinematic ankle model (Figure 3-1). The system identification problem is solved via a two-level optimization approach. Given the current guess for the 12 parameters (i.e., the model structure is fixed), the lower-level analysis function (or sub-optimization) adjusts the generalized coordinates in the model (i.e., the model configuration is varied) so as to minimize the 3D coordinate errors between modeled and experimental surface markers. During each lower-level optimization, the optimal solution from the previous time frame is used to seed the subsequent time frame. The upper-level optimization adjusts the 12 parameters defining the joint structure so as to

minimize the cost function calculated by the lower-level optimization over all time frames. In the present study, this is achieved using the BFGS gradient-based optimizer.

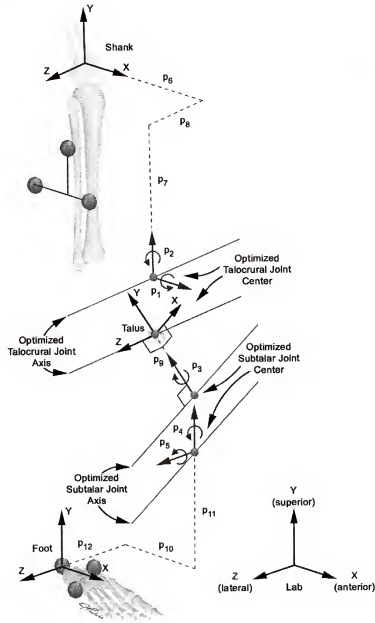


Figure 3-1. Kinematic ankle joint model used in analysis function. The model is three dimensional, possesses eight degrees of freedom (six for the position and orientation of the shank segment, and two for the talocrural and subtalar joints), and requires 12 parameters (p_1 through p_{12}) to define the positions and orientations of the joint axes in the shank, talus, and foot segment coordinate systems.

In mathematical form, the above system identification optimization problem can be stated as follows:

$$f(x) = \min_p \sum_{t=1}^{nf} e(p, t) \quad (3-3)$$

with

$$e(p, t) = \min_q \sum_{i=1}^{nm} \sum_{j=1}^3 (a_{ij}(t) - b_{ij}(p, q))^2 \quad (3-4)$$

where Equation 3-3 is the cost function for the upper-level optimization. This equation is a function of patient-specific model parameters p and is evaluated for each of nf recorded time frames. Equation 3-4 represents the cost function for the lower-level optimization and uses a nonlinear least square algorithm to adjust the model's DOFs q to minimize errors between experimentally measured marker coordinates a and model-predicted marker coordinates b , where nm is the number of markers whose three coordinates are used to calculate errors. Hence, the sum of the squares of 3D marker coordinate errors obtained from lower-level optimization of every time frame produces the cost function for the upper-level optimization.

Table 3-1. Number of function evaluations and total execution time (wall-clock time) for sequential optimization. Execution time for function evaluations in gradient calculations and line searches takes the majority of execution time (99.97%) of sequential optimization. Function evaluations for line searches take 23.85% and gradient calculations take 76.12% of the total execution time.

	Gradient calculations	Line searches	Update Hessian matrix / Determine search direction	Total
Number of function evaluations	648	203	-	851
Execution time (seconds)	32831	10285	14	43131

Before developing parallel algorithms, performance of a sequential optimization was evaluated to investigate various possible decomposition methods. Fifty time frames of numerically generated surface marker data from previously reported isolated ankle motion experiments performed with three markers on the foot and shank were used as inputs to the optimization [20]. This approach made it possible to verify that the optimizer recovered the known correct optimal solution. An initial guess was used that was far from the known solution but still within anatomically realistic bounds. All performance results were measured on a Linux-based PC cluster (1.33GHz Athlons each with 256MB memory on a 100Mbps switched Fast Ethernet network) in the High-performance Computing & Simulation Research (HCS) Laboratory at the University of Florida. Total execution time for gradient-based optimization includes time for gradient calculation function evaluations, line search function evaluations, and other optimization processes (Hessian matrix updating, search direction determination) [24,25]. Table 3-1 shows the number of function evaluations for gradient calculations, line searches, and total sequential execution time. Sequential optimization for this problem requires over 850 function evaluations at a cost of 12 hours of wall clock time, with over 76% of that time due to gradient calculation function evaluations.

3.2.3 Description of decomposition methods

Three decomposition methods are developed (one coarse-grained, one fine-grained and one medium-grained) based on the evaluation of sequential optimization. The coarse-grained method decomposes an upper-level gradient-based optimization algorithm. The fine-grained method decomposes a lower-level analysis function. The medium-grained method decomposes both levels simultaneously.

3.2.3.1 Gradient calculation decomposition

A general approach for decomposing a gradient-based optimization is to evenly distribute the gradient calculation function evaluations to different processors. Since there are no data dependencies involved in these function evaluations, this method is commonly used for parallel gradient-based optimization algorithms [6,18-20]. The same decomposition method was implemented for the unconstrained BFGS algorithm using VisualDOC API functions [23] with the Message Passing Interface (MPI) [12,13]. A master-slave paradigm was used where the master processor had responsibility for running the optimizer and distributing function evaluations.

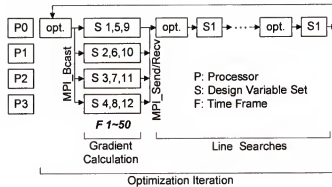


Figure 3-2. Block diagram for gradient calculation decomposition in a 4-processor system. Parallelization is performed only for the optimizer gradient calculations. Each processor evaluates the analysis function for all 50 time frames of data. For gradient calculations, each processor evaluates only 3 of 12 design variable sets, while for line searches, only the master processor (P0) repeatedly evaluates 1 design variable set.

Gradient calculation decomposition can be visualized using a block diagram for a sample 4-processor system, where each processor performs a finite difference evaluation for 3 design variable sets (Figure 3-2). Since the sample system identification problem has 12 design variables, 12 independent function evaluations can be distributed to the available processors. For each gradient calculation, the master processor broadcasts a

design variable set (input data) and optimization settings to the slave processors and gathers the results upon completion using MPI Send/Receive functions. Broadcast and send/receive communications occur only once for each iteration of the upper-level optimization. After completion of the gradient calculations, only the master processor performs line searches while the slave processors sit idle. Since a line search is an inherently sequential process, it was not considered for parallelization in this decomposition method.

Expected parallel performance of this decomposition method depends heavily on the percentage of the time spent on line searches. The parallel portion is not the entire upper-level optimization but only three quarters of it, since gradient calculations take 76% of sequential execution time. The percentage of time spent on gradient calculations can be different for other gradient-based optimization algorithms or other problems. Even though gradient calculation time fraction is expected to increase with the number of design variables, line searches are still required and cannot be parallelized easily with this decomposition method.

3.2.3.2 Analysis function decomposition

Analysis function decomposition is based on knowledge of the independent nature of the sub-optimizations. Execution time for function evaluations takes the majority (>99%) of the total sequential execution time. According to Amdahl's law, performance of a parallel algorithm is limited by the portions of the algorithm that cannot be parallelized [11]. Since a larger percentage of analysis function computations can be parallelized than can optimizer computations, this decomposition method has the potential to produce better parallel performance.

Parallel decomposition at this level required minor modifications to the sample analysis function. As noted earlier, the analysis function performs a separate sub-optimization for each time frame of data, where the solution from one time frame is used to seed the initial guess for the next time frame. To create a parallel version of the same functionality, the analysis function was modified such that the seed for the sub-optimizations was re-initialized to zero after every five time frames. This modification permitted parallelization in blocks of five time frames, had little effect on computational speed, and produced no changes in optimal cost function or design variable values to within the precision of numerical truncation errors.

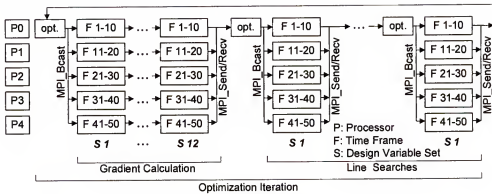


Figure 3-3. Block diagram for analysis function decomposition in a 5-processor system. Parallelization is performed only for the analysis function. Each processor evaluates the analysis function for only 10 of 50 time frames of data. For gradient calculations, each processor evaluates all 12 design variable sets, while for line searches, all 5 processors repeatedly evaluate 1 design variable set.

Analysis function decomposition can be visualized using a block diagram for a sample 5-processor system, where each processor performs sub-optimizations on 10 consecutive time frames (Figure 3-3). During gradient calculations, the master processor broadcasts the same 12 design variable sets to all slave processors, and each slave processor executes lower-level optimizations on 2 consecutive sets of 5 time frames for

each design variable set. To reduce communication frequency, the slave processors do not send their responses to the master processor after completion of each design variable set. Instead, each slave processor sequentially executes lower-level optimizations for all 12 design variable sets, stores the results, and then sends them as a group to the master processor using MPI Send/Receive functions. After receiving all responses from the slave processors, the master processor builds up the total response for each design variable set from the 10 time frames analyzed by each slave processor. This approach is used for line searches (1 design variable set) as well as gradient calculations (12 design variable set).

Communication overhead is larger for this approach than with gradient calculation decomposition because of frequent communication and large message size. The communication frequency is approximately 6 times larger than that of gradient calculation decomposition, since in this case communication occurs before and after line searches as well as gradient calculations. The communication message size is 12 times larger when a 10-processor system is used for analysis function decomposition compared to a 12-processor system for gradient calculation decomposition. However, the benefit is finer granularity than with gradient calculation decomposition, since each processor executes only a portion of the analysis function rather than the entire function. As a result, every slave processor performs virtually the same amount of work for gradient calculations and line searches over the course of the solution.

3.2.3.3 Combined decomposition

Combined decomposition is a hybrid approach that combines the benefits of gradient calculation and analysis function decomposition. This approach can be visualized using a block diagram for a sample 10-processor system, where each processor

performs sub-optimizations on 10 consecutive time frames for 6 design variable sets (Figure 3-4). During gradient calculations, available slave processors are divided into groups (e.g., 2 groups of 5 processors in a 10-processor system) for gradient calculation decomposition, where processors in each group execute assigned lower-level optimizations exactly as in analysis function decomposition. The number of groups depends on the number of design variables, and the number of processors in each group depends on the number of time frames used in the lower-level optimizations. However, during line searches, combined decomposition is identical to analysis function decomposition, since line searches were not parallelized. During that phase, only one group of processors is used and the remaining processors sit idle until the line search is completed.

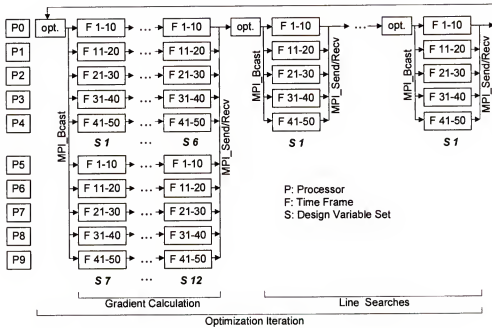


Figure 3-4. Block diagram for combined decomposition in a 10-processor system.

Parallelization is performed for both the gradient calculations and the analysis function. Each processor evaluates the analysis function for only 10 of 50 time frames of data. For gradient calculations, each processor evaluates 6 of 12 design variable sets, while for line searches, only 5 of 10 processors repeatedly evaluate 1 design variable set.

This decomposition method also has larger communication overhead than does gradient calculation decomposition due to high communication frequency and large message size. However, it has less communication overhead than does analysis function decomposition because of relatively smaller message size and a reduced number of processors involved in communication during line searches. The communication frequency is the same as analysis function decomposition, as communication occurs before and after gradient calculations and line searches. The communication message size is half that of analysis function decomposition during gradient calculations, since the 12 design variable sets are divided into 2 groups. In addition, only half of the slave processors are involved in communication during line searches. This decomposition method has the advantage of using a large number of processors even though only a small number of design variables are used in the analysis function.

3.2.4 Description of evaluation metrics

To compare the performance of the three proposed decomposition methods, we performed sequential and parallel optimizations using the ankle joint system identification problem. The same initial guess and optimization parameters, such as finite difference step size and convergence criteria, were used in all optimizations. This method ensured that the sequential and parallel implementations produced identical results. For each decomposition method, the number of processors (less than or equal to 20) was selected to achieve an even workload distribution: 4-, 6-, and 12-processor systems for gradient calculation decomposition, 5- and 10-processor systems for analysis function decomposition, and 10- and 20-processor systems for combined decomposition. The maximum number of processors that can be used for gradient calculation

decomposition is limited by the number of design variables, putting an upper bound on performance improvements with this method.

For each optimization approach, performance was quantified using three measures: total execution time, speedup (the ratio of sequential execution time to parallel execution time), and parallel efficiency (the ratio of speedup to the number of processors). Speedup and parallel efficiency are particularly important metrics. Ideally, speedup should equal the number of processors (i.e., use of n processors should decrease execution time by a factor of n). In practice, this outcome is almost never achieved, and how close the speedup is to the number of processors is an indication of the parallel efficiency. Parallel efficiencies that decrease rapidly as the number of processors increases indicate increasing overhead due to communication time, synchronization time, or the parallel algorithm itself. The best use of computational resources is achieved when the speedup is close to the number of processors and thus the parallel efficiency is close to 100%.

3.3 Results

Parallel decomposition at one or both levels resulted in significant performance improvements compared to sequential optimization of the ankle joint system identification problem. Total execution time decreased as the number of processors increased for each of the parallel decomposition methods (Figure 3-5a). This decrease was approximately linear with the number of processors for analysis function and combined decomposition but not for gradient calculation decomposition. For comparable numbers of processors, analysis function decomposition was the fastest and gradient calculation decomposition the slowest. Consistent with these observations, speedup was closest to the number of processors for analysis function decomposition and farthest from the number of processors for gradient calculation decomposition (Figure 3-5b).

Consequently, parallel efficiency was better than 90% for analysis function decomposition with 5 and 10 processors while it dropped from 59% to 28% for gradient calculation decomposition as the number of processors increased from 4 to 12 (Figure 3-5c). All decomposition methods exhibited at least some decrease in parallel efficiency with increasing number of processors.

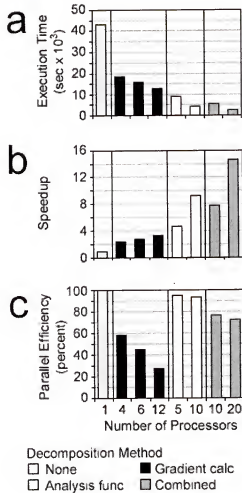


Figure 3-5. Performance metrics for the parallel decomposition methods as a function of the number of processors. (a) Total execution time due to optimizer computations, communication and synchronization overhead, and analysis function computations. (b) Speedup - the ratio of sequential execution time to parallel execution time. (c) Parallel efficiency - the ratio of speedup to the number of processors. For a fixed number of processors, analysis function decomposition provided the best performance and gradient calculation decomposition the worst.

Breakdown of execution time revealed that analysis function computations were the major contributor (Table 3-2). For all three decomposition methods, computation time varied in a manner similar to total execution time. Optimizer time, which is the time required by the master processor to perform optimization algorithm tasks such as updating the Hessian matrix and determining the search direction, was negligible and constant across all methods. Overhead for communication and synchronization time increased with the number of processors and was largest for analysis function decomposition and smallest for gradient calculation decomposition. However, it was still small compared to computation time, never accounting for more than 7% of total execution time.

Table 3-2. Optimizer, overhead, and computation time breakdown (in seconds) for the three parallel algorithms as a function of the number of processors. Overhead includes both communication and synchronization time.

Parallel algorithms	Gradient calculation decomposition			Analysis function decomposition		Combined decomposition	
Number of processors	4	6	12	5	10	10	20
Optimizer (seconds)	14	14	14	14	14	14	14
Overhead (seconds)	25	29	86	77	155	119	198
Computation (seconds)	18468	15740	12993	8966	4466	5481	2737

3.4 Discussion

This study has presented three approaches for decomposing biomechanical optimization problems for parallel processing. The first decomposes optimizer gradient calculations for distribution to multiple processors, the second decomposes the analysis function (typically a kinematic or dynamic simulation) called repeatedly by the optimizer to calculate the cost function and constraints, and the third decomposes gradient

calculations and the analysis function simultaneously. These approaches were demonstrated on an ankle joint system identification problem using a standard gradient-based optimizer. Though gradient calculation decomposition is the most common [6,18-20], analysis function decomposition resulted in the most computationally efficient parallel algorithm. Somewhat surprisingly, combined decomposition performed slightly worse than analysis function decomposition for a comparable number of processors, indicating that the additional work to parallelize the optimizer was not helpful for this particular problem. Since parallelization effort is typically spent on the optimizer rather than the analysis function, these results suggest that future parallel biomechanical optimization efforts should consider focusing on the analysis function instead.

The main reason that gradient calculation decomposition performed the worst was our inability to parallelize line searches. During a line search, all slave processors sat idle while the master processor performed repeated analysis function calls to determine how far to move in the calculated search direction. This load imbalance grew as the number of processors increased. Since line searches required a significant amount of fixed execution time, total execution time could not decrease linearly with increasing number of processors, causing parallel efficiency to drop. Parallelization of line searches would improve performance significantly [26,27]. For analysis functions with a larger number of design variables, such as Anderson and Pandy's three-dimensional jumping simulation with 432 design variables [18], the effects of line search inefficiencies may be diminished relative to the gradient calculation computation time. Thus, even with gradient calculation decomposition, speedup and parallel efficiency may improve substantially for large-scale problems. The other limitation of this method is that the maximum number of

processors is bounded by the number of design variables, which is why only 12 processors were used for this method. This limitation would also be eliminated for large-scale problems with hundreds of design variables, where low overhead due to communication and synchronization would become a more significant advantage.

Analysis function decomposition performed the best primarily because of its finer workload granularity as the number of processors increased. Finer granularity equates to lower likelihood of load imbalances that hurts parallel efficiency. This decomposition method is not limited by the characteristics of the optimization algorithm or the number of design variables and is able to parallelize the largest percentage of the total computations. These benefits outweighed the larger overhead caused by frequent communication and synchronization with large message sizes, since the overhead was still small compared to the required computation time. Despite these benefits, analysis function decomposition only makes sense if the analysis function involves computationally costly calculations such as sub-optimizations or solution of large systems of equations. Otherwise communication and synchronization overhead may negate the performance benefits of lower-level parallelization. Furthermore, while gradient calculation decomposition can be reused without modifications to different optimization problems, analysis function decomposition is specific to a particular problem. However, once the analysis function decomposition has been implemented, the problem can be reused with different types of optimization algorithm when biomechanical researchers wanted to use several optimization algorithms to find the better solution.

Combined decomposition showed performance trends similar to analysis function decomposition since it includes the gradient calculation decomposition method in its formulation. However, it exhibited slightly worse performance than analysis function decomposition due to the presence of idle processors during line searches. Though an even workload distribution could be obtained by distributing 5 rather than 10 time frames to each processor during line searches, the extra effort to parallelize at both levels would not produce significant additional performance gains.

The results reported in our study are specific to gradient-based optimizers. Non-gradient methods, such as global population-based genetic [3], simulated annealing [2,3], and particle swarm [10,20-22] algorithms, do not have line searches. For those methods, parallel decomposition of gradient calculations is replaced with parallel decomposition of sampling within the design space. Since more points in design space can be sampled than the number of design variables, the number of processors used for sampling decomposition is not limited by the number of design variables, though other factors may limit parallel efficiency. For example, since the particle swarm algorithm generally works best with 20 particles (i.e., 20 sample points in the design space) regardless of the number of design variables, use of more than 20 processors in a parallel implementation would not produce further performance gains [28]. The drawback of global optimizers is the significantly greater computation time required to obtain a solution. Thus, for problems where a good initial guess can be obtained, gradient-based optimizers still possess a distinct advantage in terms of computation time.

Our results are also specific to a particular kinematic analysis function. Parallel decomposition of our analysis function was facilitated by the fact that sub-optimization of

each time frame could be performed independently from other time frames. In addition to kinematic simulations, analysis functions involving inverse dynamics simulations possess this desirable characteristic. In contrast, forward dynamics simulations of human movement are difficult to parallelize since numerical integration is a sequential process. Wider availability of parallel numerical integrators would be valuable for such applications [29]. Alternatively, forward dynamic optimization problems can sometimes be reformulated as equivalent inverse dynamic optimization problems [30,31], thereby permitting parallelization of the analysis function directly (see Appendix). This concept opens up a variety of parallelization possibilities that yet to be investigated in the biomechanics community. It is likely that other biomechanical optimization problems of interest could also be reformulated to permit analysis function decomposition.

3.5 Summary

In summary, this study has presented three parallel algorithms for biomechanical optimization. The algorithms were applied to a biomechanical optimization problem involving system identification of a kinematic ankle joint model. A gradient-based optimizer was used with an analysis function that determined the optimal alignment of the model with experimental movement data given the current guess at the model parameters. Parallelization of optimizer gradient calculations resulted in the worst performance for a fixed number of processors while parallelization of the analysis function resulted in the best performance. Thus, parallelization of the optimizer may not always be the most computationally efficient choice. Significant performance gains for gradient-based optimizers would be obtained by parallelizing line searches as well. An interesting direction for future research would be to apply analysis function decomposition to other computationally intensive biomechanical optimization problems

using gradient and non-gradient parallel optimization algorithms. The approach to use the analysis function decomposition on a wide variety of biomechanical optimization problems enables the biomechanical researchers to perform complex simulations with many optimization algorithms (or parallel optimization algorithms). Especially, the movement prediction problems are computationally intensive and difficult to parallelize the analysis function due to the numerical integration. Developing analysis function decomposition for movement prediction problems be a challenging problem. To reduce idle processor time, dynamic load balancing scheme can be employed to improve parallel performance in heterogeneous environments with different processor speeds. Meanwhile, asynchronous approach for parallel optimization algorithms, which is an algorithmic approach, could be another option to enhance the parallel performance. Furthermore, as the number of processors increases, so does the likelihood of processor failures. Thus, fault-tolerant algorithms for parallel optimization will need to be developed and implemented.

CHAPTER 4

PARALLEL ALGORITHMS FOR PARTICLE SWARM OPTIMIZATION

In this chapter, we introduce a parallel asynchronous PSO (PAPSO) algorithm to enhance computational efficiency. The performance of the algorithm is evaluated in homogeneous and heterogeneous computing environments for small- to medium-scale analytical test problems and a medium-scale biomechanical test problem. The rest of this chapter is organized as follows. Section 4.1 introduces an overview of this chapter. Section 4.2 presents background of the particle swarm global search method and the proposed parallel asynchronous optimization algorithm. Section 4.3 provides a description of sample test problems (analytical problems and biomechanical optimization problem) and evaluation metrics. The results and their performances are analyzed in Section 4.4 and 4.5 respectively. Finally, Section 4.6 presents a brief summary.

4.1 Introduction

Particle swarm optimization (PSO) is a stochastic optimization method often used to solve complex engineering problems such as structural and biomechanical optimizations [e.g.,10,20-22,32-34]. Similar to evolutionary optimization methods, PSO is a derivative-free, population-based global search algorithm. PSO uses a population of solutions, called particles, which fly through the search space with directed velocity vectors to find better solutions. These velocity vectors have stochastic components and are dynamically adjusted based on historical and inter-particle information.

Recent advances in computer and network technologies have led to the development of a variety of parallel optimization algorithms, most of which are

synchronous in nature. A synchronous optimization algorithm is one that requires a synchronization point at the end of optimizer iteration before continuing to the next iteration. For example, parallel population-based algorithms such as genetic [3,10,35], simulated annealing [36,37], pattern search [38], and particle swarm [22] methods typically wait for completion of all function evaluations by the population before determining new sets of design variables. Similarly, parallel gradient-based algorithms [5,6,18,19] require gradients to be calculated for all design variables before determining a new search direction.

Parallel synchronous optimization algorithms work best when three conditions are met. First, the optimization has total and undivided access to a homogeneous cluster of computers without interruptions from other users. Second, the analysis function takes a constant amount of time to evaluate of any set of design variables throughout the optimization. Third, the number of parallel tasks can be equally distributed among the available processors. If any of these three conditions is not met, the parallel optimization algorithm will not make the most efficient use of the available computational resources. Since large clusters are often heterogeneous with multiple users, the first condition is frequently violated. Furthermore, the computational cost to evaluate a complex analysis function is often variable during an optimization [10]. Finally, since most parallel optimization algorithms have coarse granularity, the number of parallel tasks to be assigned to the available processors is usually not large enough to balance the workload. All three factors can lead to load imbalance problems that significantly degrade parallel performance.

Parallel optimization algorithms employing an asynchronous approach are a potential solution to the load imbalance problem. The asynchronous approach does not need a synchronization point to determine a new search direction or new sets of design variables. Thus, the optimization can proceed to the next iteration without waiting for the completion of all function evaluations from the current iteration. Despite this advantage, few parallel optimization algorithms have been asynchronous. Exceptions include specific parallel asynchronous implementations of Newton or quasi-Newton [39,40], pattern search [41], genetic [42,43], and simulated annealing [44] algorithms. Only recently have preliminary implementations of parallel asynchronous PSO algorithms been investigated [45,46].

In this study, we propose and evaluate a parallel asynchronous particle swarm optimization (PAPSO) algorithm that dynamically adjusts the workload assigned to each processor, thereby making efficient use of all available processors in a heterogeneous cluster. The robustness, performance, and parallel efficiency of our PAPSO algorithm are compared to that of a parallel synchronous PSO (PSPSO) algorithm [22] using a suite of analytical test problems and a benchmark biomechanical test problem. Parallel performance of both algorithms is evaluated on homogeneous and heterogeneous Linux clusters using problems that require constant and variable amounts of computation time per function evaluation.

4.2 Particle Swarm Optimization

Particle swarm global optimization is a class of derivative-free, population-based computational methods introduced by Kennedy and Eberhart in 1995 [47]. Particles (design points) are distributed throughout the design space and their positions and velocities are modified based on knowledge of the best solution found thus far by each

particle in the “swarm.” Attraction towards the best-found solution occurs stochastically and uses dynamically-adjusted particle velocities. Particle positions (Equation 4-1) and velocities (Equation 4-2) are updated as shown below:

$$x_{k+1}^i = x_k^i + v_{k+1}^i \quad (4-1)$$

$$v_{k+1}^i = wv_k^i + c_1r_1(p_k^i - x_k^i) + c_2r_2(p_k^g - x_k^i) \quad (4-2)$$

where x_k^i represents the current position of particle i in design space and subscript k indicates a (unit) pseudo-time increment. The point p_k^i is the best-found position of particle i up to time step k and represents the cognitive contribution to the search velocity v_k^i . The point p_k^g is the global best-found position among all particles in the swarm up to time step k and forms the social contribution to the velocity vector. Random numbers r_1 and r_2 are uniformly distributed in the interval $[0, 1]$, while c_1 and c_2 are the cognitive and social scaling parameters, respectively (see [10] for further details). The following sections describe the synchronous and asynchronous design of sequential PSO algorithm and the proposed parallel asynchronous particle swarm optimization algorithm.

4.2.1 Synchronous vs. asynchronous design

While several modifications to the original PSO algorithm have been made to increase robustness and computational throughput [48-52], one of the key issues is whether a synchronous or asynchronous approach is used to update particle positions and velocities. The sequential synchronous PSO algorithm updates all particle velocities and positions at the end of every optimization iteration (Figure 4-1). In contrast, the sequential asynchronous PSO algorithm updates particle positions and velocities continuously based on currently available information (Figure 4-2). The difference between the two methods is similar to the difference by Jacobi (synchronous) and Gauss-

Seidel (asynchronous) methods for solving linear systems of equations. Carlisle and Dozier [53] evaluated the performance of both methods on analytical test problems using a sequential PSO algorithm that did not include dynamically-adjusted particle velocities. They concluded that the asynchronous approach is generally less costly computationally than is the synchronous approach based on the number of function evaluations required to achieve convergence. However, the performance of the two update methods were problem dependent and the differences were not significant.

```

Initialize Optimization
    Initialize algorithm constants
    Randomly initialize all particle positions and velocities
Perform Optimization
    For  $k = 1$ , number of iterations
        For  $i = 1$ , number of particles
            Evaluate analysis function  $f(x_k^i)$ 
        End
        Check convergence
        Update  $p_k^i$ ,  $p_k^g$ , and particle positions and velocities  $x_{k+1}^i$ ,  $v_{k+1}^i$ ;
    End
Report Results
  
```

Figure 4-1. Pseudo-code for a sequential synchronous PSO algorithm.

To date, the only parallel implementation of a PSO algorithm has been synchronous [22]. The rationale for the synchronous approach was to maintain consistency between sequential and parallel implementations, thereby avoiding alteration of the convergence characteristics of the algorithm. Thus, parallel synchronous PSO obtains exactly the same final solution as sequential synchronous PSO if the sequence of random numbers

generated by the algorithm (i.e., r_1 and r_2 in Equation 4-2) are constrained to be the same. Because the parallel performance of a synchronous implementation can suffer from load imbalance, PSPSO works best when there is no heterogeneity in either the computing environment or evaluation time for the analysis function and when the number of particles is an integer multiple of the number of processors.

```

Initialize Optimization
    Initialize algorithm constants
    Randomly initialize all particle positions and velocities
Perform Optimization
    For  $k = 1$ , number of iterations
        For  $i = 1$ , number of particles
            Evaluate analysis function  $f(x_k^i)$ 
            Check convergence
            Update  $p_k^i$ ,  $p_k^r$ , and particle positions and velocities  $x_{k+1}^i$ ,  $v_{k+1}^i$ 
        End
    End
Report Results

```

Figure 4-2. Pseudo-code for a sequential asynchronous PSO algorithm.

4.2.2 Parallel asynchronous design

Current parallel computing environments and optimization tasks make it difficult to achieve the ideal conditions required for high parallel efficiency with PSPSO. To overcome this problem, we propose an adaptive concurrent strategy, called parallel asynchronous particle swarm optimization (PAPSO), which can increase parallel performance significantly. In addition to parallelization, this strategy involves one significant modification to the sequential asynchronous PSO algorithm. Since the

updated velocity and position vectors of particle i are calculated using the best-found position p_k^i and the global best-found position p_k^g up to iteration k , the order of particle function evaluations affects the outcome of the optimization. However, the PSO algorithm does not restrict the order in which particle function evaluations are performed. Consequently, we permit the particle order to change continuously depending on the speed with which each processor completes its function evaluations, thereby eliminating the use of iteration counter k in our modified APSO algorithm.

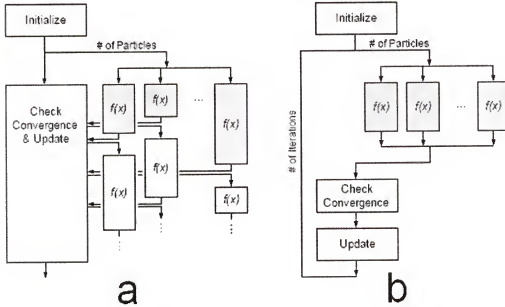


Figure 4-3. Block diagrams for (a) parallel asynchronous and (b) parallel synchronous PSO algorithms. Gray boxes indicate first set of particles evaluated by each algorithm code for a sequential synchronous PSO algorithm.

The proposed PAPSO approach can be understood at a high level by comparing a block diagram of its algorithm with that of a PPSO algorithm [22] (Figure 4-3). Our PAPSO algorithm updates particle positions and velocities continuously based on currently available information (Figure 4-3a). In contrast, PPSO updates particle positions and velocities at the end of each optimizer iteration using global

synchronization (Figure 4-3b). Thus, while PSPSO uses static load balancing to determine processor workload at compile time, PAPSO uses dynamic load balancing to determine processor workload during run time, thereby reducing load imbalance. The remainder of this section describes the key features incorporated into the proposed PAPSO algorithm.

Our PAPSO design follows a master/slave paradigm. The master processor holds the queue of particles ready to send to slave processors and performs all decision-making processes such as velocity/position updates and convergence checks. It does not perform any function evaluations. The slave processors repeatedly evaluate the analysis function using the particles assigned to them. The tasks performed by the master and slave processors are as follows:

- Master processor
 1. Initializes all optimization parameters and particle positions and velocities
 2. Holds a queue of particles for slave processors to evaluate
 3. Updates particle positions and velocities based on currently available information p^i, p^g
 4. Sends the position x^i of the next particle in the queue to an available slave processor
 5. Receives cost function values from slave processors
 6. Checks convergence
- Slave processor
 1. Receives a particle position from the master processor
 2. Evaluates the analysis function $f(x^i)$ at the given particle position x^i
 3. Sends a cost function value to the master processor

Once the initialization step has been performed by the master processor, particles are sent to the slave processors to evaluate the analysis function. Consequently, the initial step of the optimization is identical to that of the PPSO algorithm.

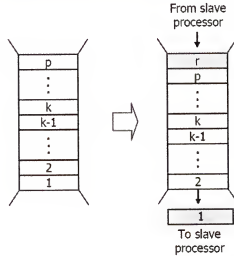


Figure 4-4. Block diagram for first-in-first-out centralized task queue with p particles on a k -processor system. After receiving the cost function value and corresponding particle number r from a slave processor, the master processor stores the particle number at the end of the queue, updates the position of the first particle, and sends the first particle back to idle slave processor for evaluation. The particle order changes depending on the speed with which each processor completes its function evaluations.

After initialization, the PPSO algorithm uses a first-in-first-out centralized task queue to determine the order in which particles are sent to the slave processors (Figure 4-4). Whenever a slave processor completes a function evaluation, it returns the cost function value and corresponding particle number to the master processor, which places the particle number at the end of the task queue. Since the order varies in which particles report their results, randomness in the particle order occurs. Thus, unlike synchronous PSO, sequential and parallel implementations of asynchronous PSO will not produce identical results. Once a particle reaches the front of the task queue, the master processor updates its position and sends it to the next available slave processor. Even with

heterogeneity in tasks and/or computational resources, the task queue ensures that each particle performs approximately the same number of function evaluations over the course of an optimization.

Communication between master and slave processors is achieved using a point-to-point communication scheme implemented with the Message Passing Interface [12,13]. Since there is no global synchronization in PAPSO, communication time is hidden within the computation time of the slave processors. Because the master processor can communicate with only one slave processor at a time, each slave processor remains idle for a short period of time while waiting to connect to the master processor after completing a function evaluation. However, this idle time is typically negligible compared to the computation time required for each function evaluation. The point-to-point communication scheme is inefficient when the computational cost is the same for all function evaluations but is very efficient when the computational cost varies for different function evaluations.

4.3 Sample Optimization Problems

The robustness, performance, and parallel efficiency of our PAPSO algorithm are compared to that of a parallel synchronous PSO using a suite of analytical test problems and a benchmark biomechanical test problem. The following sections describe the characteristics of the sample analytical problems and biomechanical test problem.

4.3.1 Analytical test problems

To evaluate the performance of the proposed PAPSO algorithm, a suite of difficult analytical problems with known solutions [3,10,22,53] was used as test cases. Each problem in the suite was evaluated using the proposed PAPSO algorithm and a previously published PSPSO algorithm [22]. Each run was terminated based on a pre-

defined number of function evaluations for the particular problem being solved. A detailed description of the four analytical test problems can be found in Soest et al. [3], Schutte et al. [10,22], and Carlisle and Dozier [53]. The following is a brief description of the analytical test problems:

- H1: This simple 2-dimensional function [3,22] has several local maxima and a global maximum of 2 at the coordinates (8.6998, 6.7665). 10,000 function evaluations were used for this problem. An optimization was considered to be successful when the error between true solution and optimized solution was less than 0.001 (i.e., acceptable error) [3].

$$H_1(x_1, x_2) = \frac{\sin^2\left(x_1 - \frac{x_2}{8}\right) + \sin^2\left(x_2 + \frac{x_1}{8}\right)}{d + 1} \quad x_1, x_2 \in [-100, 100] \quad (4-3)$$

$$\text{where } d = \sqrt{(x_1 - 8.6998)^2 + (x_2 - 6.7665)^2}$$

- H2: This inverted version of the F6 function used by Schaffer et al. [54] has two design variables with several local maxima around the global maximum of 1.0 at (0,0). This problem was solved using 20,000 function evaluations per optimization run, and the acceptable error was 0.001 [3].

$$H_2(x_1, x_2) = 0.5 - \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)^2} \quad x_1, x_2 \in [-100, 100] \quad (4-4)$$

- H3: This test function from Corona et al. [55] was used with dimensionality $n = 4, 8, \text{ and } 16$. The function contains a large number of local minima (on the order of 10^{4n}) with a global minimum of 0 at $|x_i| < 0.05$. The number of function evaluations used for this problem was 50,000 ($n = 4$), 100,000 ($n = 8$), and 200,000 ($n = 16$). Acceptable error was 0.001 [3].

$$H_3(x_1, \dots, x_n) = \sum_{i=1}^n \begin{cases} (t \cdot \text{sgn}(z_i) + z_i)^2 \cdot c \cdot d_i & \text{if } |x_i - z_i| < t \\ d_i \cdot x_i^2 & \text{otherwise} \end{cases} \quad (4-5)$$

$$x_i \in [-1000, 1000]$$

$$\text{where } z_i = \left\lfloor \frac{|x_i|}{s} \right\rfloor + 0.49999 \cdot \text{sgn}(x_i) \cdot s, \quad c = 0.15,$$

$$s = 0.2, t = 0.05, \text{ and } d_i = \begin{cases} 1 & i = 1, 5, 9, \dots \\ 1000 & i = 2, 6, 10, \dots \\ 10 & i = 3, 7, 11, \dots \\ 100 & i = 4, 8, 12, \dots \end{cases}$$

- H4: This test problem from Griewank [56] superimposes a high frequency sine wave on a multi-dimensional parabola. The function has a global minimum of 0 at $x_i = 0.0$, and the number of local minima increases exponentially with the number of design variables. To investigate large-scale optimization issues, problems were formulated using 32 and 64 design variables. The number of function evaluations used for this problem was 320,000 ($n = 32$) and 640,000 ($n = 64$) [22]. Acceptable error was 0.1 as recommended in Carlisle and Dozier [53].

$$H_4(x_1, \dots, x_n) = \sum_{i=1}^n \frac{x_i^2}{d} - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad x_i \in [-600, 600] \quad (4-6)$$

where $d = 4000$.

4.3.2 Biomechanical test problem

In addition to the analytical test problems, a real-life biomechanical test problem was included to evaluate PAPS performance when different function evaluations require significantly different computation times, thereby creating a large load imbalance. The benchmark problem involves determination of patient-specific parameter values that permit a three-dimensional (3D) kinematic ankle joint model to reproduce experimental movement data as closely as possible (Figure 4-5) [20]. Twelve parameters (treated as design variables) specify the fixed positions and orientations of joint axes fixed in adjacent body segments (i.e., shank, talus, and foot) within the 8 degree-of-freedom (DOF) kinematic ankle model. After specifying realistic values for the 12 joint parameters, we impose realistic motions on the joints in the model to create synthetic (i.e., computer-generated) trajectories for three surface markers fixed to the foot and lower leg.

The goal of the optimization is to find the 12 joint parameters that allow the model to reproduce the motion of the six surface markers as closely as possible.

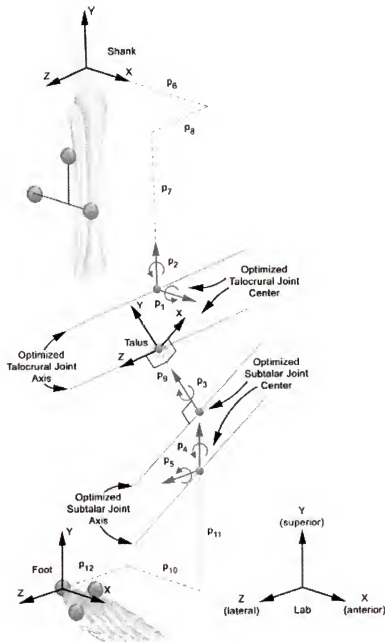


Figure 4-5. Kinematic ankle joint model used in the biomechanical test problem. The model is three dimensional, possesses eight degrees of freedom (six for the position and orientation of the shank segment and two for the ankle joint), and requires 12 parameters p_1 through p_{12} to define the positions and orientations of the joint axes fixed in the shank, talus, and foot segment coordinate systems.

This system identification problem is solved via a two-level optimization approach. Given the current guess for the 12 parameters, a lower-level optimization (or sub-optimization) adjusts the DOFs in the model so as to minimize the 3D coordinate errors between modeled and synthetic marker locations for a single time frame. The upper-level optimization adjusts the 12 parameters defining the joint structure so as to minimize the sum of the cost function values calculated by the lower-level optimization over all time frames.

In mathematical form, the above system identification optimization problem can be stated as follows:

$$f(x) = \min_p \sum_{t=1}^{nf} e(p, t) \quad (4-7)$$

with

$$e(p, t) = \min_q \sum_{i=1}^{nm} \sum_{j=1}^3 (a_{ij}(t) - b_{ij}(p, q))^2 \quad (4-8)$$

where Equation 4-7 is the cost function for the upper-level optimization. This equation is a function of patient-specific model parameters p and is evaluated for each of nf recorded time frames. Equation 4-8 represents the cost function for the lower-level optimization and uses a nonlinear least-squares algorithm to adjust the model's DOFs (q) to minimize errors between experimentally measured marker coordinates a_{ij} and model-predicted marker coordinates b_{ij} , where nm is the number of markers whose three coordinates are used to calculate errors.

4.3.3 Evaluation metrics

A variety of metrics were used to quantify the performance, robustness, and parallel efficiency of the PAPSO algorithm compared to the PSPSO algorithm. For all PSO runs

(asynchronous and synchronous), 20 particles were used unless otherwise noted, and PSO algorithm parameters were set to standard recommended values (see [10] for details). Parallel efficiency was evaluated on two test beds. The first was a homogeneous Linux cluster of 20 identical machines connected with a Gigabit Ethernet switched network, where each machine possessed a 1.33GHz Athlon processor and 256 MB of memory. The second was a group of 20 heterogeneous machines chosen from several Linux clusters (Table 4-1). Parallel efficiency is defined as the ratio of speedup to the number of processors, where speedup is the ratio of sequential execution time to parallel execution time on a homogeneous cluster. Evaluating the biomechanical test problem on a heterogeneous cluster introduced two sources of load imbalance into the evaluation process: the processors and the problem itself. Both test beds were located in the High-performance Computing and Simulation (HCS) Research Laboratory at the University of Florida, and all tests were performed without other users on the selected processors.

Table 4-1. Summary of heterogeneous computing resources used to evaluate execution time for the biomechanical test problem.

CPU Speed	CPU Type	Memory	Network	Nodes
2.4 GHz	Intel Xeon	1 GB	Gigabit Ethernet	3
1.4 GHz	AMD Opteron	1 GB	Gigabit Ethernet	3
1.3 GHz	AMD Athlon	256 MB	Gigabit Ethernet	3
1.0 GHz	Intel Pentium-III	256 MB	Fast Ethernet	3
733 MHz	Intel Pentium-III	256 MB	Fast Ethernet	3
600 MHz	Intel Pentium-III	256 MB	Fast Ethernet	3
400 MHz	Intel Pentium-II	128 MB	Fast Ethernet	2

For the analytical test problems, evaluation metrics were calculated from 100 optimization runs using different initial guesses. Robustness was measured as the fraction of runs that converged to within the specified tolerance of the known optimal cost function value. Performance was quantified as the mean and standard deviation of

the number of function evaluations required to reach the optimal solution for runs that converged to the correct solution. Parallel efficiency was evaluated by running a modified version of one analytical test problem (H3 with $n = 16$) for 40,000 function evaluations on the homogeneous test bed. For this problem only, 32 particles were used with 1, 4, 8, 16, and 32 processors. Time delays of 0.5, 1, or 2 seconds were added to each function evaluation to increase the ratio of computation time to communication time. To mimic real-life variability, we increased the time delay for each function evaluation by up to 0%, 20%, or 50% of its nominal value using uniform random sampling. For example, a 2-second time delay randomly increased by up to 50% would produce delays ranging from 2 to 3 seconds (Table 4-2).

Table 4-2. Range of computation time delays [min, max] for each function evaluation of problem H3 with $n = 16$. Times are uniformly distributed within the indicated interval.

Variation (%)	Delay Time (seconds)		
	0.5	1	2
0	[0.5, 0.5]	[1.0, 1.0]	[2.0, 2.0]
20	[0.5, 0.6]	[1.0, 1.2]	[2.0, 2.4]
50	[0.5, 0.75]	[1.0, 1.5]	[2.0, 3.0]

For the biomechanical test problem, evaluation metrics were calculated from 10 optimization runs using 10 different initial guesses (PAPSO and PSPSO) and the same initial guess 10 times (PAPSO), thereby creating three cases for comparison. Due to the random nature of particle order in PAPSO, reusing the same initial guess will not produce the same results. Robustness was reported as the mean final cost function value and the associated root-mean-square (RMS) error in marker distance and joint parameter values. Performance was evaluated by plotting the average convergence history for each of the three cases. Parallel performance was quantified for both the homogeneous and the

heterogeneous test bed. For the homogeneous test bed, execution time, speedup, and parallel efficiency were calculated for 1, 5, 10, and 20 processor systems. For the heterogeneous test bed, speedup and parallel efficiency cannot be calculated due to heterogeneity in the computational resources, so only a comparison of total execution time for the homogeneous versus heterogeneous test beds using 20 processors was reported.

4.4 Results

Overall, PAPSO algorithm performance and robustness were comparable to that of the PSPSO algorithm for the analytical test problems. For 100 independent runs, both algorithms converged to the correct solution 100% of the time for problems H1, H3 with $n = 4$ and 8, and H4 with $n = 32$ (Table 4-3). The fraction of unsuccessful runs for problems H2, H3 with $n = 16$, and H4 with $n = 32$ was approximately the same for both algorithms. Convergence speed for both algorithms was statistically the same ($p > 0.05$ based on Student's t -tests) on all problems except H2, where PAPSO performed slightly better (Table 4-4).

Table 4-3. Fraction of 100 optimization runs that successfully reached the correct solution to within the specified tolerance (see text) for each analytical test problem. n indicates the number of design variables used for each problem.

Parallel PSO Algorithm	Analytical Test Problem						
	H1	H2	H3			H4	
	($n=2$)	($n=2$)	($n=4$)	($n=8$)	($n=16$)	($n=32$)	($n=64$)
Synchronous	1.00	0.61	1.00	1.00	0.82	1.00	0.96
Asynchronous	1.00	0.61	1.00	1.00	0.82	1.00	0.91

For the analytical test problem, PAPSO and PSPSO parallel efficiency exhibited different responses to increased number of processors, delay time, and delay time variation (Figure 4-6). PAPSO parallel efficiency was worse than that of PSPSO for

small numbers of processors but generally better for large numbers of processors. While PSPSO parallel efficiency decreased approximately linearly with increasing number of processors to as low as 76% for 32 processors, PAPSO parallel efficiency increased nonlinearly from only 75% for 4 processors to over 90% for 16 processors and 92% for 32 processors. These trends were sensitive to delay time and its variation for PSPSO but not PAPSO, with the sensitivity increasing with increasing number of processors. PSPSO parallel efficiency decreased when delay time was decreased from 2 to 0.5 seconds (i.e., decreased computation time) and when delay time variation was increased from 0 to 50%. For 32 processors, each 20% change in delay time variation decreased PSPSO parallel efficiency by about 5%.

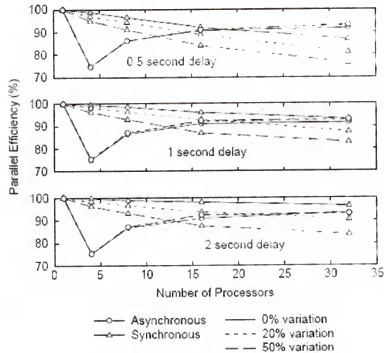


Figure 4-6. Parallel efficiency for the parallel asynchronous (circles) and parallel synchronous (triangles) PSO algorithms as a function of number of processors, computational delay time (0.5, 1, or 2 seconds), and computational delay time variation (0, 20, or 50%). The optimization used 32 particles for analytical test problem H3 with $n = 16$ and was performed for 40,000 function evaluations on a homogeneous cluster.

Table 4-4. Mean number of function evaluations required to reach the final solution for optimization runs that converged to the correct solution for each analytical test problem. Standard deviations are indicated in parentheses.

Parallel PSO Algorithm	Analytical Test Problem						
	H1 (n=2)	H2 (n=2)	H3 (n=4) (n=8) (n=16)			H4 (n=32) (n=64)	
Synchronous	3,396 (376)	6,243 (4,280)	4,687 (452)	7,549 (701)	24,918 (5,108)	15,403 (1,294)	29,843 (2,790)
Asynchronous	3,495 (385)	6,594 (4,171)	4,560 (444)	7,577 (576)	25,036 (6,330)	15,427 (1,449)	29,949 (2,553)

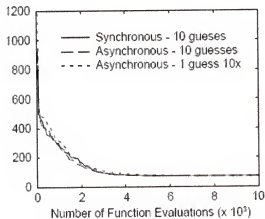


Figure 4-7. Mean convergence history plots over 10,000 function evaluations for the biomechanical test problem obtained with the parallel asynchronous and parallel synchronous PSO algorithms. Each set of 10 optimizations used either 10 different sets of initial guesses (synchronous and asynchronous) or 1 set of initial guesses 10 times (asynchronous). Refer to Table 4-5 for corresponding quantitative results.

For the biomechanical test problem, PAPS0 again demonstrated comparable performance and robustness to that of PSPSO. After 10,000 function evaluations, final cost function values, RMS marker distance errors, RMS orientation parameter errors, and RMS position parameter errors were statistically the same ($p > 0.05$ based on pair-wise Student's t-tests) for PSPSO with 10 sets of initial guesses, PAPS0 with 10 sets of initial guesses, and PAPS0 with 1 set of initial guesses used 10 times (Table 4-5). All three approaches successfully recovered ankle joint parameter values consistent with

previously reported results [10]. Mean convergence history plots as a function of number of function evaluations also exhibited similar convergence speed for all three approaches (Figure 4-7).

Table 4-5. Mean final cost function values and associated RMS marker distance and joint parameter errors after 10,000 function evaluations obtained with the parallel synchronous and asynchronous PSO algorithms. Standard deviations are indicated in parentheses. Each set of 10 optimizations used either 10 different sets of initial guesses (synchronous and asynchronous) or 1 set of initial guesses 10 times (asynchronous).

Parallel PSO Algorithm	Initial Guesses	Cost Function Value	RMS Error		
			Marker Distances (mm)	Orientation Parameters (deg)	Position Parameters (mm)
Synchronous	10	70.40	5.49	4.85	2.40
		(1.18)	(0.04)	(3.00)	(1.44)
Asynchronous	10	69.92	5.47	3.19	2.39
		(0.84)	(0.03)	(2.28)	(1.45)
Asynchronous	1	70.65	5.50	4.89	2.51
		(1.01)	(0.04)	(2.47)	(1.34)

Parallel performance on the biomechanical test problem was very different between the two PSO algorithms. For the homogeneous test bed, parallel performance showed similar trends to those found with the analytical test problem (Figure 4-8). Total execution time decreased and speedup increased for both algorithms as the number of processors increased. However, PAPSO execution time and speedup were worse (higher and lower, respectively) than that of PSPSO on the 5-processor system but better on the 10- and 20-processor systems (e.g., 7 hours vs. 10 hours on the 20-processor system).

In contrast, parallel efficiency increased for PAPSO with increasing number of processors beyond one but decreased for PSPSO. Similar to execution time and speedup, parallel efficiency was worse for PAPSO than for PSPSO on the 5-processor system but better on the 10- and 20-processors systems. The parallel efficiency gap between the two

algorithms increased with number of processors, reaching almost 30% for the 20-processor system. For the heterogeneous test bed with 20 processors, execution time was approximately 3.5 times less for PAPS than for PAPS (Figure 4-9). Compared to the homogeneous environment, execution time increased for both algorithms, but the increase was much smaller for PAPS.

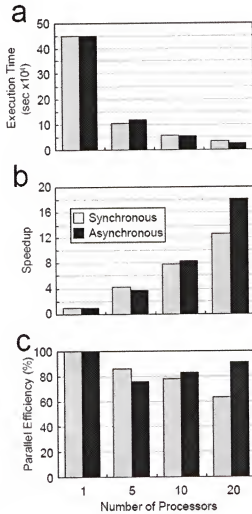


Figure 4-8. Execution time (a), speedup (b), and parallel efficiency (c) for parallel asynchronous and parallel synchronous PSO algorithms for the biomechanical test problem in a homogeneous computing environment.

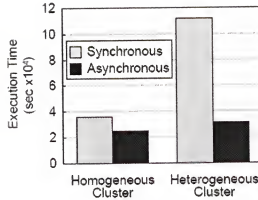


Figure 4-9. Execution time for parallel asynchronous and parallel synchronous PSO algorithms for the biomechanical test problem in a heterogeneous computing environment.

4.5 Discussion

This study presented an asynchronous parallel algorithm for particle swarm optimization. The algorithm was evaluated using a set of analytical test problems and a biomechanical test problem involving system identification of a 3D kinematic ankle joint model. Speedup and parallel efficiency results were excellent when there was heterogeneity in computing resources or the optimization problem itself. For problems utilizing a large number of processors, small computation-to-communication time ratio, or large variability in computation time per function evaluation, the proposed PAPSO algorithm exhibits excellent parallel performance along with optimization performance and robustness comparable to that of PSPSO.

Parallel efficiency of the PAPSO algorithm exhibited an unusual trend as a function of the number of processors. For most parallel algorithms, parallel efficiency decreases as the number of processor increases because of increased overhead caused primarily by inter-processor communication. This observation explains why PSPSO parallel performance decreased with an increasing number of processors. However, PAPSO

exhibited the opposite trend. For the 5-processor system, PSPSO used all five processors to evaluate the analysis function, whereas PAPSO used only four processors since the master processor does not perform function evaluations. Thus, the master processor was idle much of the time in the 5-processor system, reducing speedup and parallel efficiency. Since the negative impact of one idle processor decreases with increasing number of processors, PAPSO parallel performance improved rapidly as the number of processors was increased, asymptotically approaching 93%. In addition, communication overhead for the PAPSO algorithm was smaller than for the PSPSO algorithm, since PAPSO uses point-to-point communication between the master processor and each slave processor.

The modified analytical test problem showed that both magnitude and variability of computation time for function evaluations affect parallel performance of PSPSO but not PAPSO. An increase in computation time through added constant time delay resulted in an increased computation-to-communication time ratio, improving parallel efficiency for PSPSO. In contrast, increasing the variability in computation time through time delay variations degraded PSPSO parallel efficiency due to load imbalance. Consequently, the parallel performance of PSPSO is relatively good when the computation-to-communication time ratio is high and when the computation time per function evaluation is relatively constant for any set of design variables used during the optimization. PSPSO may also exhibit poorer parallel performance when the parallel optimization involves a large number of processors due to increased communication overhead.

There are three primary limitations to the proposed PAPSO algorithm. First, parallel efficiency is poor for low numbers of processors, as discussed above. Second, two runs starting with the same particle locations in design space will not produce

identical answers, though the same is true for PSPSO if the random number sequence generated by the algorithm is not fixed. Third, the degree of algorithm parallelism is limited by the number of particles, a problem shared with PSPSO. The maximum number of processors that can be used for PSPSO is equal to the number of particles and the number of particles plus one for PAPS0, putting an upper bound on performance improvements. However, PAPS0 can achieve good parallel efficiency for any number of processors less than the maximum, whereas PSPSO can only achieve good parallel efficiency when the number of particles can be distributed in equal groups to the available processors. For both algorithms, if more processors are available than the number of particles, parallelization of the analysis function as well (i.e., parallelization on two levels) could be used to achieve improved scalability [5].

4.6 Summary

In summary, this study introduced a parallel asynchronous particle swarm optimization algorithm that exhibits good parallel performance for large numbers of processors as well as good optimization robustness and performance. Since PAPS0 incorporates a dynamic load balancing scheme, parallel performance is dramatically increased for (1) heterogeneous computing environments, (2) user-loaded computing environments, and (3) problems producing run-time load variations. For a sample problem where the computational cost of the analysis function varied for different sets of design variables, PAPS0 achieved better than 90% parallel efficiency on a homogeneous 20-processor system while PSPSO achieved only 62%. For the same problem run on a heterogeneous 20-processor system, PAPS0 finished a given number of function evaluations 3.5 times faster than did PSPSO. However, PAPS0 performance is worse than that of PSPSO for small numbers of processors and/or if the time for each function

evaluation is large (compared to communication time) and constant throughout the optimization. Overall, PAPSO provides a valuable option for parallel global optimization under heterogeneous computing conditions.

CHAPTER 5

EVALUATION OF PARALLEL GLOBAL SEARCH METHOD FOR LARGE-SCALE MOVEMENT OPTIMIZATION PROBLEMS

In this chapter, we evaluate the parallel asynchronous PSO algorithm for large-scale movement optimization problems. The performance of the algorithm is compared to the commonly used gradient-based optimization algorithm. The rest of this chapter is organized as follows. Section 5.1 introduces an overview of this chapter. Section 5.2 presents background of the parallel particle swarm global search algorithm and a description of the large-scale optimization problem formulation. The results and their performance are analyzed in Sections 5.3 and 5.4 respectively. Finally, Section 5.5 presents a brief summary.

5.1 Introduction

Biomechanical optimization problems frequently use optimization methods to solve system identification or movement prediction problems [1,17-20]. Dynamic optimization problems, such as human movement simulations, are computationally intensive when the simulation involves a complex musculoskeletal model: a large number of controls and degrees of freedom resulting in optimization problems with many local minima. For these large-scale problems, optimization algorithms generally have a high computational cost since they iteratively evaluate the cost function and constraints to obtain a converged solution. Even with fast-converging gradient-based optimization algorithms, the complexities of present-day biomechanical models can require thousands of function evaluations to achieve convergence [2,5].

To increase the throughput of the optimization, several parallel optimization algorithms have been developed and evaluated for biomechanical optimization problems. Parallel algorithms for gradient-based optimization algorithms have been used [5,6,18,19] as well as global search methods such as genetic [3], simulated annealing [36], and particle swarm [10,20,22] optimization algorithms. However, only gradient-based optimization algorithms have been used for large-scale biomechanical optimization problems. In a previous study, Anderson and Pandey [6,19] solved a large-scale gait simulation using a parallel gradient-based optimization algorithm. Other parallel global search methods have been used for small- to medium-scale biomechanical optimization problems [3,10,20,22,36].

This study evaluates a particle swarm global search method to solve large-scale biomechanics optimization problems. Algorithm performance was compared to that of a commonly used gradient-based optimization algorithm. A gait movement prediction problem was developed as a large-scale sample problem. The problem involves inverse dynamic optimization of a three-dimensional, 27 degree-of-freedom (DOF), full body gait model tuned to gait data collected from a single knee osteoarthritis (OA) patient. The optimizations reduced the peak knee adduction torque subject to several reality constraints and predicted novel motions for which experimental data were not available. The specific goal of this study was to evaluate a parallel asynchronous particle swarm optimization algorithm [45,46] for large-scale movement optimization problems.

5.2 Methodology

To evaluate the performance of the PAPSO algorithm for a large-scale biomechanical test problem, we have used the nonlinear least squares method for comparison purposes. The following sections describe the parallel particle swarm

optimization algorithm, nonlinear least squares method, and large-scale biomechanical test problems.

5.2.1 Parallel particle swarm optimization algorithms

Particle swarm global optimization (PSO) is a class of derivative-free, population-based computational methods, introduced by Kennedy and Eberhart in 1995 [47]. In the original PSO, particles (design points) are distributed throughout the design space and their positions and velocities are modified based on the knowledge of the best solution found thus far by each particle in the “swarm.” Attraction towards the best-found solution occurs stochastically and uses dynamically adjusted particle velocities. Particle positions (Equation 5-1) and velocities (Equation 5-2) are updated as shown below:

$$x_{k+1}^i = x_k^i + v_{k+1}^i \quad (5-1)$$

$$v_{k+1}^i = wv_k^i + c_1r_1(p_k^i - x_k^i) + c_2r_2(p_k^g - x_k^i) \quad (5-2)$$

where x_k^i represents the current position of particle i in design space and subscript k indicates a (unit) pseudo-time increment. The point p_k^i is the best-found position of particle i up to time step k and represents the cognitive contribution to the search velocity v_k^i . The point p_k^g is the global best-found position among all particles in the swarm up to time step k and forms the social contribution to the velocity vector. Random numbers r_1 and r_2 are uniformly distributed in the interval $[0, 1]$, while c_1 and c_2 are the cognitive and social scaling parameters, respectively (see [22] for further details).

Recently, two parallel algorithms for PSO were developed and evaluated on a biomechanical system identification problem and a typical aircraft wing design problem: parallel synchronous PSO (PSPSO) [22] and parallel asynchronous PSO (PAPSO) [45,46]. The main difference between those parallel algorithms is the update method for

particle position, x_{k+1}^i , and velocity vectors, v_{k+1}^i . In PPSO, particles (design points) are distributed to processors, and then each processor evaluates one (or more) analysis function(s) corresponding to the assigned particles. Once evaluations of the analysis function are completed for each processor, the master processor gathers fitness values of all evaluations using global synchronization and communication. Based on these fitness values from all particle evaluations, the master processor determines the best-found positions (p_k^i, p_k^g) and updates particle velocities and positions (Equations 5-1, 5-2) for the next iteration. Meanwhile in PPSO, the master performs all decision-making processes such as velocity/position updates and convergence checks, and the slave processors only evaluate the analysis function corresponding to the particles assigned to them. Whenever a slave processor completes a function evaluation, it sends the fitness value and the corresponding particle number to the master processor. The master receives the fitness value, and updates the particle's position and velocity based on the currently available best solution that is stored in the master processor. The previous study [45] showed that the PPSO algorithm is recommended for problems where the computation time for all evaluation time is constant and where the computation-to-communication ratio is relatively high and when a small number of processors is involved. The PPSO algorithm works best when there is either heterogeneity in computing resources or tasks.

5.2.2 Nonlinear least squares method

For comparison purposes, the Levenberg-Marquardt nonlinear least squares (LM-NLS) optimization algorithm [57] available in Matlab (The Mathworks, Natick, MA) was applied to the test problem. The Levenberg-Marquardt method is a popular gradient-

based algorithm that provides a numerical solution to the problem of minimizing a sum of squares of errors in nonlinear functions. The algorithm requires numerical or analytical evaluations of the gradients and tends to converge only if the initial guess is already close to the optimal solution.

5.2.3 Large-scale biomechanical test problem

A real-life large-scale biomechanical problem was used to evaluate both optimization algorithms. The benchmark problem involves prediction of gait movement changes that permit a three-dimensional gait model to reduce the left knee adduction torque as much as possible subject to several reality constraints. A three-dimensional (3D) gait model was developed using Autolev (Online Dynamics, Inc., Sunnyvale, CA). The model possesses 27 degrees-of-freedom (DOFs) composed of gimbal (3 DOFs), universal (2 DOFs), and pin (1 DOF) joints (Figure 5-1). The position and orientation of the pelvis in the laboratory reference frame was defined by three translational and three rotational DOFs. For the lower-body joints, each hip was modeled as a gimbal joint, each knee as a pin joint, and each ankle as two non-intersecting and non-orthogonal pin joints. For the upper body, the back was modeled as a gimbal joint, each shoulder as a universal joint, and each elbow as a pin joint. The ground reaction forces and torques calculated using the force plate electrical centers were treated as unknowns to be determined during periods when the foot was known to be in contact with the ground [58].

Joint body and segment parameters in the model were tuned to match the gait data using the Levenberg-Marquardt nonlinear least squares optimization algorithm [59]. Experimental gait data were collected from a single knee OA patient (male, age 39 years, height 170cm, mass 70kg, alignment 5° varus with grade 2 medial osteoarthritis in both

knees). One complete gait cycle (left heel strike to right heel strike) with clean surface marker and ground reaction data was selected for use in the optimization study.

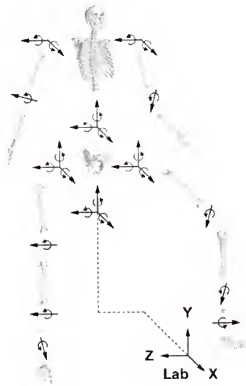


Figure 5-1. Schematic of the 27 degree-of-freedom full-body gait model used to predict novel gait motions that reduce the peak knee adduction torque. All joints are traditional engineering joints (gimbal, universal, pin) with the exception of the ground to pelvis joint which possesses 6 degrees of freedom. Parameters defining the locations and orientations of joints in the body segments and the masses, mass centers, and moments of inertia of the body segments were tuned to gait data collected from a single knee osteoarthritis patient.

While forward dynamic optimization is commonly used for movement prediction problems [18], we used an inverse dynamic optimization approach to predict novel motions. This method places design variables on the joint motion rather than joint torque curves. By allowing the optimization algorithm to change the input motions, novel gait movements can be predicted that also satisfy joint torque constraints. Motion and ground reaction curves were parameterized as a function of time using a combination of polynomial and Fourier terms [60]. The design variables were defined as associated

coefficients of the parameterized curves. To match the gait data accurately, a cubic polynomial with eight Fourier harmonics (i.e., 20 coefficients) was found to be adequate.

Two problems were formulated to evaluate the two optimization algorithms. One was a toe-out gait optimization and the other a modified gait optimization. The toe-out gait optimization problem predicted the peak knee adduction torque when the foot-progression angle was increased by 15 deg. Previous experimental studies [61,62] have shown that the position of the foot affects the adduction torque at the knee. The modified gait optimization predicted novel gait changes that could minimize the peak knee adduction torque [63,64].

For both optimization problem formulations, the motion and ground reaction curves were allowed to vary: pelvis translations, all back, pelvis, hip, knee, and ankle rotations, and all ground reaction forces and torques (33 curves in all for a total of 660 design variables). Shoulder and elbow rotations were prescribed to match the experimental data. Ground reaction forces and torques were set to zero for time frames when the foot was known to be off the floor.

Using these design variables, inverse dynamic optimizations were performed to predict novel gait motion and the peak knee adduction torque by the PAPSO and nonlinear least squares algorithms. The cost function for toe-out gait optimization minimizes the foot-progression angle difference between experimental gait and additional +15 deg offset subject to several reality constraints via a penalty method (Equation 5-3). The cost function for gait movement modification minimizes the left knee adduction torque subject to several reality constraints via a penalty method.

$$\begin{aligned}
\min \sum_{f=1}^{nframes} \left\{ \Delta T_{LAdd}^2 + \sum_{s=1}^2 \left(\sum_{j=1}^3 (\Delta T_{Hip}^2)_{fsj} + (\Delta T_{Knee}^2)_{fs} + \sum_{j=1}^2 (\Delta T_{Ankle}^2)_{fsj} + \sum_{j=1}^2 (\Delta CoP^2)_{fsj} \right) + \right. \\
\left. \sum_{s=1}^2 \left(\sum_{j=1}^3 (\Delta q_{Hip}^2)_{fsj} + (\Delta q_{Knee}^2)_{fs} + \sum_{j=1}^2 (\Delta q_{Ankle}^2)_{fsj} \right) + \sum_{j=1}^6 (\Delta q_{Pelvis}^2)_{fj} \right. \\
\left. w_{Trunk} \sum_{j=1}^3 (\Delta q_{Trunk}^2)_{fj} + w_{Pelvis} \sum_{j=1}^6 (\Delta T_{Pelvis}^2)_{fj} + w_{FootPath} \sum_{s=1}^2 \left(\sum_{j=1}^6 (\Delta q_{Foot}^2)_{fsj} \right) \right\} \quad (5-3)
\end{aligned}$$

where $nframes$ is the number of time frames that are used in the problem (100 time frames), f is the time frame, j is the translational or rotational joint axis number, and s is side. The weights for reality constraints are $w_{Trunk}=w_{Pelvis}=w_{FootPath}=10$ found by trial and error. T_{LAdd} is the left knee adduction torque to minimize during the optimization. For the toe-out gait optimization, T_{Ladd} was removed. Constraints are the change in hip flexion/extension, abduction/adduction, or inter/external rotation torque and angle (ΔT_{Hip} , Δq_{Hip}), the change in knee flexion/extension torque and angle (ΔT_{Knee} , Δq_{Knee}), the change in ankle flexion/extension or inversion/eversion torque and angle (ΔT_{Ankle} , Δq_{Ankle}), the change in center of pressure x and z translation (ΔCoP), the change in pelvis x, y, z translation/rotation (Δq_{Pelvis}), the change in trunk x, y or z rotation (Δq_{Trunk}), the change in external pelvis x, y or z force or torque (ΔT_{Pelvis}), and the change in foot x, y or z translation or rotation (Δq_{Foot}) away from its experimental value measured with respect to the lab frame.

5.2.4 Optimization setup

For all parallel asynchronous PSO runs, 20 particles were used with standard recommended parameters (see [10] for details). For Matlab's Levenberg-Marquardt nonlinear least squares method, default settings were used. Optimization results were reported as the mean final fitness value, mean percent reduction in the left knee adduction torque and RMS errors from five runs using PPSO and one run using the nonlinear least

squares method. For PSO, each run started from the different initial guess (particle position) with one of the particles assigned to the experimental values. Due to the nature of the complex design space and many local minima, starting from the different initial guesses may not produce the same results even for a global search method. Multiple independent runs are useful for evaluating optimization robustness. For the nonlinear least squares algorithm, one run was started from the experimental values. The optimizations were evaluated on a homogeneous Linux cluster of 20 identical machines connected with a Gigabit Ethernet switched network, where each machine possessed a 1.33GHz Athlon processor and 256 MB of memory. This cluster is located in the High-performance Computing and Simulation (HCS) Research Laboratory at the University of Florida.

5.3 Results

Overall, the nonlinear least squares optimization algorithm successfully converged for both optimization problems with significant reductions in the peak knee adduction torque during left leg stance (Figure 5-2). In contrast, the PAPSO algorithm converged to an improved solution only for the toe-out gait optimization.

For both problems, the mean final fitness value for PAPSO was larger than that of nonlinear least squares method. For the percent reduction in left knee adduction torque, the nonlinear least squares method showed a larger reduction than that of PAPSO for both problems. On average, the nonlinear least squares method reduced the second peak by 32.2% for the toe-out gait optimization and the first and second peaks by 32.4% and 29.4%, respectively, for the gait modification problem. The PAPSO algorithm reduced the second peak by 9.3% for the toe-out gait optimization (Table 5-1).

Table 5-1. Mean for final fitness value and percent reduction on two peaks of the left knee abduction/adduction torque from the five runs of PAPSO and one run of LM-NLS method. Since the PAPSO algorithm did not converge to the solution, only the nonlinear least squares algorithm prediction was shown for gait modification problem.

Problem	Optimizer	Final fitness	Adduction Torque Reduction (%)	
			First Peak	Second Peak
Toe-out optimization	PAPSO	170,503	3.5	9.3
	LM-NLS	66,687	2.9	32.2
Gait modification	PAPSO	7.55e+6	-	-
	LM-NLS	23,571	32.4	29.4

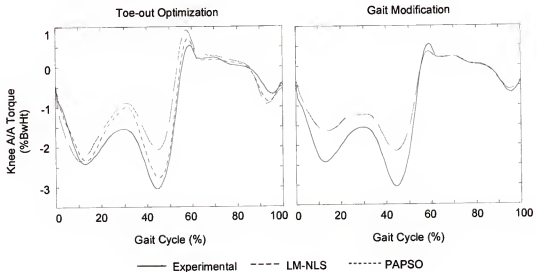


Figure 5-2. Left knee abduction/adduction (A/A) torque curves calculated from the experimental gait data (solid line) and predicted by two optimizations: PAPSO and nonlinear least squares method (LM-NLS). Adduction torque is negative. Left column contains optimization results generated with the toe-out gait optimization, while right column contains optimization results with gait modification. Since the PAPSO algorithm did not converge to the solution, only the nonlinear least squares algorithm prediction was shown for gait modification.

For three reality constraints (foot path translation/rotation, trunk orientations, and pelvis residual load errors), the nonlinear least squares method successfully reduced RMS errors within 2 mm or deg and 4 N or Nm for both problems. The PAPSO algorithm reduced RMS errors of the same magnitude only for the toe-out gait optimization

problem. Even though an additional 15 deg foot-progression angle offset was required for the toe-out gait optimization problem, the PAPSO algorithm only achieved 10 deg.

The toe-out optimization predicted large kinematic changes in hip inter/external rotation to increase the foot-progression angle. The nonlinear least squares solution produced more lateral center of pressure (CoP) than did the PAPSO solution (Figure 5-4). The gait modification optimization predicted kinematic changes that slightly decreased the hip abduction/adduction angle resulting in a medial shift of the left knee center. The center of pressure showed slightly more lateral movement than in the experimental data (Figure 5-4).

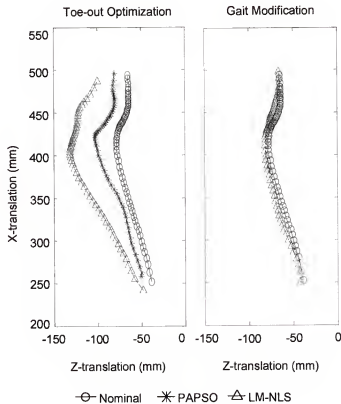


Figure 5-3. Left center of pressure (CoP) trajectories in pelvis progression frame. Left column contains optimization results generated with the toe-out gait optimization, while right column contains optimization results with the gait modification.

5.4 Discussion

This study evaluated a parallel asynchronous particle swarm optimization algorithm using a large-scale movement optimization problem: three-dimensional, 27 degree-of-freedom (DOF), full body gait model tuned to gait data collected from a single knee osteoarthritis (OA) patient. The optimizations reduced the peak knee adduction torque subject to several reality constraints and predicted novel motions for which experimental data were not available.

By increasing additional foot-progression angle subject to several reality constraints, both the PAPSO and nonlinear least squares algorithm reduced the second peak in the knee adduction torque curve (9% for PAPSO and 32.2% for nonlinear least squares method). In addition, the nonlinear least squares method reduced both peaks in the knee adduction torque curve (32.4% for the first peak and 29.4% for the second peak) for the gait modification problem.

The PAPSO algorithm exhibited worse performance than that of the nonlinear least squares method. For both problems, the performance of the nonlinear least squares method was better than that of the PAPSO algorithm in terms of the final fitness value and percent reduction in the knee adduction torque. The results indicated that the PAPSO algorithm is not applicable to these large-scale biomechanical problem formulations even though the PSO algorithm has shown good performance for small- to medium-scale problems with standard parameters [10,22]. The high complexity of design space may require parameter tuning for the PAPSO algorithm. Even though gradient-based optimization algorithms such as nonlinear least squares require fine tuning of finite difference step size before running an optimization [10], a good initial guess increases the chances of finding good solution for such large-scale problems. In addition to finite

difference step size tuning, the nonlinear least square method requires scaling of the design variables to improve accuracy of the gradient calculations. Scaling the design variables to experimental data enables the gradient-based optimization algorithms to perform relative forward differencing rather than absolute forward differencing when calculating the numerical gradients.

Using experimental motion and ground reaction forces/torques as an initial guess assisted both algorithms in finding their solutions. Neither algorithm converged starting from a random initial guess. In contrast, PSO only needs one of the initial particle positions to be set to match the experimental data, since attraction towards the best-found solution occurs stochastically for the PSO algorithms during the optimization.

The toe-out gait optimization predicted that increasing the foot-progression angle would reduce the second peak of knee adduction torque curve as stated in previous experimental studies [61,62]. For example, such was the case when toe-out angle was changed without modifying stance width measured at the heel as in our model. Manal et al. showed 40% reduction in the second peak with 15 deg increase in foot-progression angle [62]. Our toe-out gait optimization achieved 32.2% reduction in the second peak and slight change in the first peak (2.9%). The difference of the percent reduction is within the standard deviation reported by Manal et al. [62].

5.5 Summary

In summary, this study evaluated a PAPSO algorithm for large-scale human movement optimization by comparison with a nonlinear least squares method. Even though the parallel asynchronous PSO algorithm has performed well for small- to medium-scale problems, it may need parameter tuning to solve large-scale problems. In

addition, the optimization results were consistent with an experimental study on the effect of foot-progression angle on knee adduction torque.

CHAPTER 6

CONCLUSIONS

In this research, parallel algorithms for biomechanical optimization problems were presented and evaluated to meet the intensive computational complexity. The adaptive asynchronous parallel optimization approach has also been introduced and applied to the parallel particle swarm algorithm that is promising to achieve high parallel efficiency and reliability. In addition, the parallel asynchronous particle swarm optimization algorithm was applied and evaluated for a large-scale human movement optimization problem.

Recent biomechanical optimization problems have been required to include complex musculoskeletal systems so that the biomechanics researchers can accurately analyze human movement. The complex musculoskeletal system increases the computational cost of evaluating the cost function for the optimization. In addition, this complexity also increases the optimization complexity in finding the optimum solution. Although the performance of single-processor computers has vastly increased in recent years, computation time can still be a limiting factor. Parallel processing is, therefore, essential.

We first developed parallel algorithms to meet the substantial computational requirements of the biomechanical optimization problem using a gradient-based optimization algorithm. These algorithms are based on domain decomposition methods and statistically scheduled to reduce inter-processor communication and other forms of parallelization overhead. The parallel performance of the algorithm was examined in terms of execution time, speedup, and parallel efficiency on a cluster of computers.

Among parallel algorithms, the analysis function decomposition method shows the most promising parallel performance for the given optimization algorithm and problem formulation even though the gradient calculation decomposition method has been commonly used. In addition, the combined decomposition method provides a large increase in degree of parallelism where the problem has a small to medium number of design variables. The results of this research renders new insight to biomechanics researchers that the analysis function decomposition will be a promising decomposition method in providing the best parallel performance.

We extensively proposed and investigated a parallel asynchronous optimization algorithm for the particle swarm global search method. The parallel asynchronous particle swarm optimization algorithm exhibits good parallel performance for a large numbers of processors as well as good optimization robustness and performance. Since the proposed parallel algorithm incorporates a dynamic load balancing scheme, parallel performance is dramatically increased for (1) heterogeneous computing environments, (2) user-loaded computing environments, and (3) problems producing run-time load variations. For a sample problem where the computational cost of the analysis function varied for different sets of design variables, PAPSO achieved better than 90% parallel efficiency on a homogeneous 20-processor system while PSPSO achieved only 62%. For the same problem run on a heterogeneous 20-processor system, the proposed PAPSO finished a given number of function evaluations 3.5 times faster than did PSPSO. However, PAPSO performance is worse than that of PSPSO for small numbers of processors and/or if the time for each function evaluation is large (compared to communication time) and constant throughout the optimization. Overall, PAPSO

provides a valuable option for parallel global optimization under heterogeneous computing conditions.

Finally, we evaluated the parallel asynchronous particle swarm optimization algorithm in solving large-scale movement optimization problems. The performance of optimization was compared to that of a commonly used gradient-based optimization algorithm. As a large-scale biomechanical optimization, the movement prediction problem was developed. The problem involves inverse dynamic optimization of a three-dimensional, 27 degree-of-freedom (DOF), full body gait model tuned to gait data collected from a single knee osteoarthritis (OA) patient. The optimizations minimized the peak knee adduction torque subject to several reality constraints and predicted the novel motion. The optimization results indicated that gradient-based optimization algorithms may perform well for highly constrained problems, meanwhile global search methods such as particle swarm algorithm perform well for relatively low constrained problems. Even though the parallel asynchronous PSO algorithm has performed well for small- to medium-scale problems, it may need parameter tuning to solve large-scale problems. In addition, the optimization results were consistent with an experimental study on the effect of foot-progression angle on the knee adduction torque.

Our study is aimed at gaining important insights into the performance characteristics of parallel algorithms for biomechanical optimization problems. This objective requires interdisciplinary research in the areas of high-performance computing, numerical optimizations, and computational biomechanics. Novel parallel algorithms on high-performance computers connected with networks allow the parallel system to overcome significant computational complexity in biomechanical optimization problems.

Given the increasing demands for computationally sophisticated forms of dynamic optimization problems, these parallel techniques can provide a reliable and cost-effective solution.

Further research can proceed in several directions. The parallel techniques presented in this study can be extended to more enhanced and sophisticated forms of biomechanical optimization problems. A fault-tolerant scheme is needed to enhance the reliability of optimization problems. Because of the long-running computation characteristics in optimization problems, the faults that may occur in the system or processes may require restart of the whole optimization. Moreover, the reliability of parallel processing is important when a large number of processors is involved.

APPENDIX EXAMPLE OF REFORMULATED FORWARD DYNAMIC OPTIMIZATION PROBLEM

This appendix provides an example of how a forward dynamic optimization problem can be reformulated as an inverse dynamic optimization problem with parallelization of the analysis function to predict muscle activations and periodic motion when no experimental data are available. This is but one additional example of how biomechanical optimization problems can be re-cast in a form that permits analysis function parallelization. The reader is encouraged to think along similar lines for other biomechanical optimization problems of interest.

Consider a dynamic musculoskeletal model possessing n degrees of freedom controlled by m muscles, where $F_j(t)$ is the force generated by any muscle j at time t and $a_j(t)$ is the corresponding muscle activation ($0 \leq a_j(t) \leq 1$). For simplicity, assume that $F_j(t)$ can be computed by scaling the muscle's peak isometric force F_j^o by its activation [32].

$$F_j(t) = F_j^o a_j(t) \quad (\text{A-1})$$

The net muscle joint torque $T_k(t)$ at any joint k ($k=1, \dots, n$) is then a linear combination of the individual muscle forces:

$$T_k(t) = \sum_{j=1}^m r_{jk}(t) F_j(t) \quad (\text{A-2})$$

where $r_{jk}(t)$ is the moment arm of muscle j about joint k . In turn, the individual net muscle joint torques appear in the equations of motion of the multibody dynamic system (assumed to possess no closed loops for simplicity) as indicated below:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{T}(\mathbf{q}) + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) \quad (\text{A-3})$$

where,

\mathbf{q} = $n \times 1$ column matrix of generalized coordinates

$\mathbf{M}(\mathbf{q})$ = $n \times n$ system mass matrix

$\mathbf{T}(\mathbf{q})$ = $n \times 1$ column matrix due to net muscle joint torques

$\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})$ = $n \times 1$ column matrix due to centripetal and Coriolis forces

$\mathbf{G}(\mathbf{q})$ = $n \times 1$ column matrix due to gravity forces

$\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ = $n \times 1$ column matrix due applied external forces and torques.

Thus, given the activation $a_j(t)$ for each muscle j at any time t , Equations A-1 and A-2 can be used to calculate $\mathbf{T}(\mathbf{q})$ in Equation A-3.

Based on this model structure, the following forward dynamic optimization problem could be formulated to predict muscle activations that would produce a periodic motion:

$$\text{Minimize} \quad \sum_{i=1}^f \left(\sum_{j=1}^m a_j(t_i)^2 \right)$$

$$\text{subject to} \quad \ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1} [\mathbf{T}(\mathbf{q}) + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})]$$

$$\mathbf{q}(0) = \mathbf{q}(t_f)$$

$$\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}(t_f)$$

$$\begin{aligned}
T_k(t) &= \sum_{j=1}^m r_{jk}(t) F_j(t), k=1, \dots, n \\
F_j(t) &= F_j^0 a_j(t), j=1, \dots, m \\
a_j(0) &= a_j(t_f), j=1, \dots, m \\
0 \leq a_j(t) &\leq 1, j=1, \dots, m
\end{aligned} \tag{A-4}$$

by varying $q(0)$

$$\dot{q}(0)$$

$$a_j(t_l), j=1, \dots, m, \quad l=1, \dots, p$$

where f is the number of time frames analyzed and p is the number of spline nodal points used to discretize each $a_j(t)$ curve (normally $p \ll f$). This cost function has been shown by Anderson and Pandy [32] to produce muscle force estimates that are similar to those found by minimization of metabolic energy expenditure. The constraints enforce a periodic (though not necessarily symmetric) motion and bound the amplitude of the muscle activations. The design variables are the initial conditions for each DOF along with muscle activation nodal points. For any specified set of design variables, the cost function and constraints in Equation A-4 are evaluated by calling an analysis function that performs a forward dynamic simulation. Since there are typically more muscles than DOFs in the model, this formulation can result in forward dynamic optimization problems possessing hundreds of design variables [17,18,32].

The key concept for recasting this forward dynamic optimization problem as an inverse dynamic optimization problem is to swap the design variables with the predicted quantities. Instead of placing design variables on the muscle activations and predicting the resulting motion with forward dynamics, one places design variables on the motion and predicts the resulting muscle activations with inverse dynamics. The musculoskeletal

model is the same regardless of which formulation is used. The resulting inverse dynamic optimization problem is formulated as follows:

$$\begin{aligned}
 &\text{Minimize} && \sum_{i=1}^f \left(\sum_{j=1}^m a_j(t_i)^2 \right) \\
 &\text{subject to} && \mathbf{T}(\mathbf{q}) = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) - \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) \\
 &&& \mathbf{q}(0) = \mathbf{q}(t_f) \\
 &&& \dot{\mathbf{q}}(0) = \dot{\mathbf{q}}(t_f)
 \end{aligned}$$

with sub-optimization

$$\begin{aligned}
 &\text{Minimize} && \sum_{i=1}^f \left(\sum_{j=1}^m a_j(t_i)^2 \right) \\
 &\text{Subject to} && \\
 &&& \sum_{j=1}^m r_{jk}(t) F_j(t) = T_k(t), k = 1, \dots, n \\
 &&& F_j(t) = F_j^o a_j(t), j = 1, \dots, m \\
 &&& a_j(0) = a_j(t_f), j = 1, \dots, m \\
 &&& 0 \leq a_j(t) \leq 1, j = 1, \dots, m
 \end{aligned} \tag{A-5}$$

by varying $\mathbf{q}(t_l), l = 1, \dots, p$

In the analysis function for Equation A-5, numerical integration of the equations of motion (forward dynamics) is replaced with repeated algebraic solution of the equations of motion (inverse dynamics) followed by a sub-optimization (quadratic programming) to predict muscle activations from net muscle joint torques. Rather than placing design variables on the muscle activations directly, design variables are placed on spline nodal points describing the time histories of the generalized coordinates. Once the $\mathbf{q}(t)$ nodal

points are spline fit, $\dot{q}(t)$ and $\ddot{q}(t)$ can be computed at any time frame using the spline coefficients.

There are three advantages and one disadvantage of the inverse formulation compared to the forward formulation. The primary advantage is that the analysis function can be easily parallelized. Once $q(t)$, $\dot{q}(t)$, and $\ddot{q}(t)$ are computed, each time frame in the remaining calculations is independent from all other time frames, making it easy to parallelize the inverse dynamics and sub-optimization steps across time frames. Furthermore, even the spline fitting step can be parallelized across degrees of freedom. A second advantage is that potential problems related to numerical integration – namely system instability and numerical stiffness – are completely eliminated. A third advantage is a reduction in the number of design variables. The forward formulation requires $mp+2n$ design variables whereas the inverse formulation requires only np , where m is typically two to three times larger than n [17,18,32]. The one disadvantage is the computational cost of repeated sub-optimizations. However, seeding the initial guess for each time frame with the solution from the previous time frame significantly improves convergence speed. Overall computation time per function evaluation may still be less than with numerical integration depending on the number of processors available for the parallelization, the ability to obtain stable forward simulations, and the ability to integrate potentially stiff systems of equations. Note also that if net muscle joint torques rather than muscle activations are to be predicted, the sub-optimizations are eliminated from the inverse approach while the ability to parallelize the analysis function is retained, making this an attractive option for such problems [30,31].

LIST OF REFERENCES

1. Yamaguchi GT, Moran DW, Si J. A computationally efficient method for solving the redundant problem in biomechanics. *Journal of Biomechanics* 1995; 28: 999-1005.
2. Neptune RR. Optimization algorithm performance in determining optimal controls in human movement analyses. *Journal of Biomechanical Engineering* 1999; 121(2): 249-252.
3. Van Soest JK, Casius LJR. The merits of a parallel genetic algorithm in solving hard optimization problems. *Journal of Biomechanical Engineering* 2003; 125(1): 141-146.
4. Schnable RB. A view of the limitations, opportunities, and challenges in parallel nonlinear optimization. *Parallel Computing* 1995; 21: 875-905.
5. Koh BI, Reinbolt JA, Fregly BJ, George AD. Evaluation of parallel decomposition methods for biomechanical optimizations. *Computer Methods in Biomechanics and Biomedical Engineering* 2004; 7: 215-225.
6. Pandy MG. Computer modeling and simulation of human movement. *Annual Reviews of Biomedical Engineering* 2001; 3: 245-273.
7. Anderson FC, Pandy MG. A dynamic optimization solution for vertical jumping in three dimensions. *Computer Methods in Biomechanics and Biomedical Engineering* 1999; 2: 201-231.
8. Zajac FE. Muscle coordination of movement: a perspective. *Journal of Biomechanics* 1993; 26(1): 109-124.
9. Vanderplaats GN. Numerical optimization techniques for engineering design: with applications. McGraw-Hill College, 3rd Edition, New York, 1984.
10. Schutte JF, Koh BI, Reinbolt JA, Fregly BJ, Haftka RT, George AD. Evaluation of a particle swarm algorithm for biomechanical optimization. *Journal of Biomechanical Engineering* 2005; 127: 465-474.
11. Culler DE, Singh JP. Parallel computer architecture: a hardware/software approach. Morgan Kaufman Publishers Inc., San Francisco, CA, 1998.

12. Snir M, Otto S, Huss-Lederman S, Walker D, Dongarra J. MPI: the complete reference. Cambridge: MIT Press, 1996.
13. Message Passing Interface Forum. MPI: a message-passing interface standard, Technical Report CS-94-230. Computer Science Department, University of Tennessee, April 1, 1994.
14. Zomaya A. Parallel and distributed computing handbook. McGraw-Hill, New York, 1996.
15. Amdahl GM. Validity of the single-processor approach to achieving large scale computing capabilities. In Proceedings of AFIPS 30, 483-485, 1967.
16. Pandy MG, Anderson FC, Hull DG. A parameter optimization approach for the optimal control of large-scale musculoskeletal systems. *Journal of Biomechanical Engineering* 1992; 114(4): 450-460.
17. Van den Bogert J, Smith GD, Nigg BM. In vivo determination of the anatomical axes of the ankle joint complex: an optimization approach. *Journal of Biomechanics* 1994; 12: 1477-1488.
18. Anderson FC, Pandy MG. A dynamic optimization of human walking. *Journal of Biomechanical Engineering* 2001; 123: 381-390.
19. Anderson FC, Ziegler JM, Pandy MG. Application of high-performance computing to numerical simulation of human movement. *Journal of Biomechanical Engineering* 1995; 117: 155-157.
20. Reinbolt JA, Schutte JF, Fregly BJ, Koh BI, Haftka RT, George AD, Mitchell KH. Determination of patient-specific multi-joint kinematic models through two-level optimization. *Journal of Biomechanics* 2005; 38: 621-626.
21. Schutte JF, Koh BI, Reinbolt JA, Haftka RT, George AD, Fregly BJ. Scale-independent biomechanical optimization. Presented at 2003 Summer Bioengineering Conference, Sonesta Beach Resort, Key Biscayne, FL, June 25-29, 2003.
22. Schutte JF, Reinbolt JA, Fregly BJ, Haftka RT, George AD. Parallel global optimization with particle swarm algorithm. *International Journal for Numerical Methods in Engineering* 2004; 61: 2296-2315.
23. Reference Manual for VisualDOC C/C++ API. Vanderplaats Research and Development, Inc., Colorado Springs, CO, 2001.
24. Venter G, Watson BC. Exploiting parallelism in general purpose optimization. Presented at the 6th International Conference on Applications of High-Performance Computers in Engineering, Maui, Hawaii, January 26-28, 2000.

25. Venter G, Watson BC. Efficient optimization algorithms for parallel application. Presented at the 8th AIAA/USAF/NASA/ISSMO Symposium at Multidisciplinary Analysis and Optimization, Long Beach, CA, September 6-8, 2000.
26. Eldred MS, Schimel BD. Extended parallelism models for optimization on massively parallel computers Presented at the 3rd World Congress of Structural and Multidisciplinary Optimization, Buffalo, NY, May 17-21, 1999.
27. Byrd RH, Schnabel RB, Schultz GA. Parallel quasi-Newton methods for unconstrained optimization. *Mathematical Programming* 1998; 42: 273-306.
28. Schutte JF. Particle swarms in sizing and global optimization. Masters Thesis 2001; University of Pretoria, South Africa.
29. Jia Z, Leimkuhler B. A parallel multiple time-scale reversible integrator for dynamics simulation. *Future Generation Computer Systems* 2003; 19: 415-424.
30. Nagurka ML, Yen V. Fourier-based optimal control of nonlinear dynamic systems. *Journal of Dynamic Systems. Measurement and Control* 1990; 112: 17-26.
31. Lo J, Huang G, Metaxas D. Human motion planning based on recursive dynamics and optimal control techniques. *Multibody System Dynamics* 2002; 8: 433-458.
32. Venter G, Sobieszcanski-Sobieski J. Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. Presented at the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, GA, September 4-6, 2002.
33. Fourie PC, Groenwold AA. The particle swarm algorithm in topology optimization. Presented at the 4th World Congress of Structural and Multidisciplinary Optimization, Dalian, China, June 4-8, 2001.
34. Eberhart RC, Shi T. Particle swarm optimization: developments, applications and resources. Presented at the IEEE Congress on Evolutionary Computation (CEC 2001), Korea, May 27-30, 2001.
35. Pereira CMNA, Lapaa CMF. Coarse-grained parallel genetic algorithm applied to a nuclear reactor core design optimization problem. *Annals of Nuclear Energy* 2003; 30: 555-565.
36. Higginson JS, Neptune RR, Anderson FC. Simulated parallel annealing within a neighborhood for optimization of biomechanical systems. *Journal of Biomechanics* 2005, in press.
37. Hong CE, McMillin BM. Relaxing synchronization in distributed simulated annealing. *IEEE Transactions on Parallel and Distributed Systems* 1995; 6(2): 189-195.

38. White DNJ. Parallel pattern search energy minimization. *Journal of Molecular Graphics and Modelling* 1997; 15(3): 154-157.
39. Conforti D, Musmanno R. Convergence and numerical results for a parallel asynchronous quasi-Newton method. *Journal of Optimization Theory Applications* 1995; 84: 293-310.
40. Fischer H, Ritter K. An asynchronous parallel Newton method. *Mathematical Programming* 1988; 42: 363-374.
41. Hough PD, Kolda TG, Torczon VJ. Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing* 2001; 23: 134-156.
42. Hart WE, Baden S, Belew RK, Kohn S. Analysis of the numerical effects of parallelism on a parallel genetic algorithm. Presented at the 10th International Parallel Processing Symposium (IPPS '96), Honolulu, HI, April 15-19, 1996.
43. Alba E, Troya JM. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems* 2001; 17(4): 451-465.
44. Lee S, Lee KG. Synchronous and asynchronous parallel simulated annealing with multiple Markov chains. *IEEE Transactions on Parallel and Distributed Systems* 1996; 7(10): 993-1008.
45. Koh BI, Fregly BJ, George AD, Haftka RT. Parallel asynchronous particle swarm for global biomechanical optimization. Presented at the 10th International Symposium on Computer Simulation in Biomechanics, Cleveland, OH. July 28-30, 2005.
46. Venter G, Sobieszczanski-Sobieski J. A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations, Presented at the 6th World Congress on Structural and Multidisciplinary Optimization, Rio de Janeiro, May 30 - June 3, 2005.
47. Kennedy J, Eberhart RC. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, Perth, Australia 1995; 4: 1942-1948.
48. Shi Y, Eberhart RC. A modified particle swarm optimizer. In *Proceedings of IEEE International Conference on Evolutionary computation*, Anchorage, Alaska 1998; 69-73.
49. Shi Y, Eberhart RC. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, Porto VW, Saravanan N, Waagen D, Eiben AE (ed.). *Lecture Notes in Computer Science*, Berlin, Springer 1998; 1447: 591-600.
50. Eberhart RC, Shi Y. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of 2000 Congress on Evolutionary Computation*, Piscataway, NJ, IEEE Service Center, 2000; 84-88.


51. Clerc M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In Proceedings of the Congress of Evolutionary Computation, Angeline PJ, Michalewicz Z, Schoenauer M, Yao X, Zalzal A (ed.). Washington D.C., USA, IEEE Press, July, 1999; 3: 1951-1957.
52. Løvbjerg M, Rasmussen TK, Krink T. Hybrid particle swarm optimizer with breeding and subpopulations. Presented at the 3rd Genetic and Evolutionary Computation Conference (GECCO-2001), July 7-11, 2001.
53. Carlisle A, Dozier G. An off-the-shelf PSO. Presented at the Workshop on Particle Swarm Optimization, Indianapolis, April 6-7, 2001.
54. Schaffer JD, Caruana RA, Eshelman LJ, Das R. A study of control parameters affecting online performance of genetic algorithms for function optimizing. In Proceedings of the 3rd International Conference Genetic Algorithm. David JD (ed.), Morgan Kaufmann Publishers, San Mateo, California 1989; 51-60.
55. Corana A, Marchesi M, Martini C, Ridella S. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. ACM Transactions in Mathematical Software 1987; 13(3): 262-280.
56. Griewank AO. Generalized descent for global optimization. Journal of Optimization Theory Applications 1981; 34: 11-39.
57. Moré JJ. The Levenberg-Marquardt algorithm: implementation and theory. Numerical Analysis, ed. Watson GA, Lecture Notes in Mathematics 630, Springer Verlag, 1977; 105-116.
58. Popović Z. Motion transformation by physically based spacetime optimization. Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA. 1999.
59. Reinbolt JA, Fregly BJ. Creation of patient-specific dynamic models from three-dimensional movement data using optimization. Presented at the 10th International Symposium on Computer Simulation in Biomechanics, Cleveland, OH. July 28-30, 2005.
60. Nagurka ML, Yen V. Fourier-based optimal control of nonlinear dynamic systems. Journal of Dynamic Systems, Measurement, and Control 1990; 112: 17-26.
61. Andrews M, Noyes FR, Hewett TE, Andriacchi TP. Lower limb alignment and foot angle are related to stance phase knee adduction in normal subjects: a critical analysis of the reliability of gait analysis data. Journal of Bone and Joint Surgery 1996; 14: 289-295.
62. Manal K, Guo M. Foot progression angle and the knee adduction moment in individuals with medial knee osteoarthritis. Presented at the 10th Congress of the International Society of Biomechanics, Cleveland, OH. July 28-30, 2005.

63. Prodromos CC, Andriacchi TP, Galante JO. A relationship between gait and clinical changes following high tibial osteotomy. *Journal of Bone and Joint Surgery* 1985; 67A: 1188-1194.
64. Fregly BJ, Rooney KL, Reinbolt JA. Predicted gait modifications to reduce the peak knee adduction torque. Presented at the 10th Congress of the International Society of Biomechanics, Cleveland, OH., 2005.


BIOGRAPHICAL SKETCH

Byung Il Koh received the Bachelor of Science degree in electronics engineering from Ajou University, Suwon, Korea, in 1995. He worked as an assistant manager at the LG Information and Communications Co. in Seoul, Korea, from 1995 until 1999. He received the Master of Science degree in electrical and computer engineering from the University of Florida, Gainesville, FL, in 2001. He has been a graduate student pursuing the Doctor of Philosophy degree in electrical and computer engineering at the University of Florida since January 2002, and a research assistant at the High-performance Computing and Simulation (HCS) Research Laboratory since August 2000. His research was supported by the National Institutes of Health. After receiving his Ph.D. degree, he will be taking a senior researcher position at the Samsung Advanced Institute of Technology in Giheung, Korea.

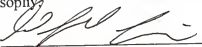
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.


Alan D. George, Chair
Professor of Electrical and Computer
Engineering


I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.


Benjamin J. Fregly, Cochair
Associate Professor of Mechanical and
Aerospace Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

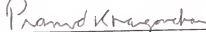

Renato J. Figueiredo
Assistant Professor of Electrical and
Computer Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.


Raphael T. Haftka
Distinguished Professor of Mechanical and
Aerospace Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

December 2005



Pramod P. Khargonekar
Dean, College of Engineering

Kenneth J. Gerhardt
Interim Dean, Graduate School